Ludwig-Maximilians-Universität München

Institut Statistik

Master Thesis

Deep Convolution Neural Networks for the Analysis of a few Medical Images

Author: Conrad Quandel

Supervisor: Prof. Dr. Volker Schmid, LMU München, Professioral Chair for Bioimaging

Date: 15. January 2018

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

München, Datum 15.01.2019

Conrad Quandel [illegible]

Name (+ Unterschrift)

Contents

List of Figures

List of Tables

1 Introduction

One of the most common reasons for blindness is diabetic retinopathy (DR) (Abràmoff et al. 2010). Visual loss can be prevented through early diagnosis, which is done via annual screenings. The manual classification of the state of DR is very time consuming for clinicians, so automated analysis and classification of these screening images can lead to early detection of DR with reliable results even in clinics without experts.

In recent years deep learning has become a popular tool for image recognition tasks not only in online challenges such as the imagenet challenge Russakovsky et al. 2015 but also in medical imaging. The problem, which is described in the first paragraph, has been investigated in a challenge on the data science competition website Kaggle in 2015 Kaggle, 2018. The results suggest that convolutional neural networks are the best tool to classify images into the different states of DR with results that could lead to automated classification tools in the future. The underlying dataset consists of 35124 images of the eye which are labelled into five classes stating the DR. One problem in medical imaging is that datasets are usually much smaller because of privacy and legal issues (Razzak et al. 2018). So one remaining question is if deep learning and in this case a convolutional neural network is still a good predictor on much smaller datasets.

In image recognition tasks the approach is widely used to pre-train a neural network using a really big dataset and retrain this network with the a much smaller dataset describing the underlying problem (Tajbakhsh et al. 2016). In this thesis a convolutional neural network called inception v3, which is pre-trained with the images of the imagenet database (Deng et al. 2009), is downloaded and retrained with the diabetic retinopathy dataset (see Chapter 3) originally used in a Kaggle challenge in 2015 (Kaggle, 2018). First a benchmark performance of the model is evaluated using all available images after splitting the dataset into training and test data. Afterwards the number of training images is decreased step by step.

The benchmark performance with an accuracy of 72.85% and a Cohen's kappa score of 0.34 is comparable to results in related papers, e.g. Butterworth et al. (2016). When the size of the training dataset is successively decreased the performance on the test data stays stable at first and decreases when approximately less than 10000 images are used. A possible threshold for the number of images that should at least be used to train the neural network is 9000 images. The requirements for this threshold are that the Cohen's kappa score is used as the performance measure and that at least 90% of the maximum Cohen's kappa score should be reached.

Problems occur because the dataset is really unbalanced with class 0 consisting of around 73% of the images. Also, computational resources limit this study, because a lot of different dataset sizes are used, some in combination with cross validation and bootstrapping, which extends the training time greatly. So other techniques such as oversampling using offline data augmentation cannot be used, because the greater dataset size would lead to increasing computational time. Nonetheless, this study shows that it is possible to train neural networks with a limited number of medical images.

Content and Structure

In the next chapter an overview over medical imaging and diabetic retinopathy will be given. Also related work is reviewed. In chapter three the dataset will be introduced and the problem is defined. The theory behind convolutional neural networks, including architecture, optimization, regularization and training is described in chapter four. In chapter five all settings that are used to train the neural networks are defined. The results will be presented in chapter six. In the last chapter a summary of this thesis, including a discussion of the results and an outlook on future work will be presented.

2 Medical Imaging

Medical Imaging implicates different techniques and applications such as computed tomography, magnetic resonance, positron emission tomography, mammography, ultrasound, X-ray, etc. (Shen et al. 2017). The analysis and interpretation of such images are mainly done by human experts. Rapid growth of databases with medical images and advances in computational resources in recent years is leading to researchers seeing automated analysis of medical images as an interesting research field (J.-G. Lee et al. 2017). Promising fields in medical image analysis are segmentation and registration of e.g. lungs, tumours, cells and membranes. In Litjens et al. (2017) the number of published papers in medical image analysis is described as exponentially growing.

Several other papers such as J.-G. Lee et al. (2017), Lakhani et al. (2018), Suzuki (2017), Razzak et al. (2018) and Jiang et al. (2010) try to give an overview over the field of deep learning in medical image analysis with practical applications . In Razzak et al. (2018) the authors provide numbers that the investment only in the medical image analysis market is expected to be 2021 as high as the investment in the whole analysis market in 2016 not limited to imaging. They also say, that "most researchers believe that within the next 15 years, deep learning based applications will take over human in performing diagnosis, predicting diseases, prescribing medicine and guiding in treatment" (Razzak et al. 2018). The biggest challenges in this sector will be privacy and legal issues because this can result in small datasets. Also deep learning is a black box which is not so easy to understand for researchers and doctors new to the field.

Most of the research in medical imaging is done with 2-dimensional data as this is the natural application where most research is done for image recognition applications. New research paper such as Kayalibay et al. (2017) or Roth et al. (2018) describe the application of three dimensional kernels to use the full information of 3-dimensional medical images.

Diabetic Retinopathy

Diabetic retinopathy (DR) is a leading cause of blindness, especially in the population of working-age adults (Hartnett et al. 2017) and the second most common cause of blindness (Abràmoff et al. 2010). It is defined as an complication of diabetes mellitus (DM). Blood vessels leak blood onto the retina, which leads to vision loss and blindness (Noronha et al. 2012). This can be caused by long term diabetes. In 2010 the number of people with DR worldwide was 126.6 million and is expected to grow to 191.0 million in 2030 according to Zheng et al. (2012). Through annual screenings and early diagnosis, visual loss and blindness can be prevented (Abràmoff et al. 2010). For annual screenings fundus imaging is commonly used. This is defined through a process where 2-D presentations of the 3-D tissues of the retina are projected onto the imaging plane (Abràmoff et al. 2010). According to Abràmoff et al. (2010) there are multiple modalities and techniques in fundus imaging. In this thesis colour fundus images are used where "the intensities represent the amount of reflected red, green and blue (RGB) wavebands" (Abràmoff et al. 2010). This technique is used because performing the acquisition of colour fundus images is cheap, non-invasive and easy (Noronha et al. 2012).

Related Work

For medical image analysis and particularly the analysis of colour fundus images many different statistical techniques exist. In recent years neural networks are the most popular technique and main focus of this work. Papers containing neural network as the prediction technique differ in how to measure the goodness of fit and usage of neural networks. In the next part an overview over different approaches and the most important papers is given.

In the mid 1990s, Gardner et al. (1996) tried to recognize vessels, exudates, haemorrhages using neural networks with just 180 images. Since then the number of publications on automated diagnosis of diabetic retinopathy using neural networks has exploded, particularly in the last two to three years. This can be explained e.g. through more publicly available data. Data from the Kaggle challenge from 2015 (see Chapter 3) and the Messidor dataset with 1200 images in four classes (Decencière et al. 2014) are often used.

Different approaches of training a neural network are used. A network can be trained from scratch (also called full training) or using pre-trained weights from a network trained on a different dataset (see also Chapter 4 or e.g. in Tajbakhsh et al. (2016)). In the latter paper the case is made that fine tuning should be used especially with smaller datasets. Still in many papers like Pratt et al. (2016), Wang et al. (2017) and Gulshan et al. (2016) full training is used. Interesting in these papers is the approach of full training, which is split into multiple stages starting with less data to pre-train weights and increasing datasets afterwards to adapt and tune weights. These approaches could be seen as a combination of full training and fine tuning.

In all papers a differentiation has to be made between classification of two or either four or five classes. In papers with 2 classes results are often published using the

Area under the ROC-curve (AUC) (Fawcett, 2006) as the performance measure and the results range from 0.94 (Ting et al. 2017), 0.97 in Gargeya et al. (2017), 0.95 in Quellec et al. (2017) to 0.99 (Gulshan et al. 2016). Here the problem of reproducibility has to be mentioned. In Voets et al. (2018) the authors try to reproduce the results of Gulshan et al. (2016) using slightly different data for training and fail to reach the same AUC.

As the main leading question of this thesis is how much a dataset can be reduced while reaching almost the same performance, paper using Fine Tuning are of main interest. In Lam et al. (2018) the Kaggle dataset is used in combination with the Messidor dataset. As both differ in the number of classes, in the Kaggle dataset the classes with severe and proliferative diabetic retinopathy are merged (see Chapter 3). The Transfer Learning approach is also called a fixed feature extractor which means that only the last layer is trained and all other weights are fixed (see Chapter 4). The Accuracy reached on two classes is around 75%, on three classes 70% and on four classes around 55%. In Alban et al. (2016) the Accuracy reached is only 42%. In the paper of Butterworth et al. (2016) an Accuracy of 76% is reported. Here Fine Tuning is approached using pretrained weights, but retraining all the weights.

Originally the dataset which is introduced in the next chapter has been used in a Kaggle challenge (Kaggle, 2018). After the challenge was finished, the winner and the runner up team published competition reports. The winner (Graham, 2015) used rather simple data preprocessing and data augmentation methods. The main work and complexity is in form of an ensemble of three neural networks and average the predicted probabilities to get a final probability distribution. Afterwards the probability distribution of the other eye and some more variables are used to train a random forest for the final predictions of each test image. The runner up team published the competition report on github (Antony et al. 2015). Interesting facts are that the convolutional neural networks is trained in three steps using smaller images to pre-train weights and retrain with larger images. Also the problem is treated as a regression problem using the mean squared error as loss function and classifying images using the thresholds 0.5, 1.5, 2.5 and 3.5. Other reports of the challenge can be found e.g. in Xu et al. (2015).

3 Data and Problem

Data

In the chapter above it is already discussed, that a lot of papers use the Kaggle or the Messidor dataset. In this thesis images from the “Diabetic Retinopathy Detection” challenge by the open internet data science website Kaggle (Kaggle, 2018, website) are used. For this challenge the dataset has been provided by a free platform for retinopathy screening called EyePACS (EYEPACS, 2018, website). These images are taken under a wide variety of imaging conditions so that images vary in size and brightness. Also, there are always pairs of images for left and right eye.

The images are labelled into 5 classes which describe the state of diabetic retinopathy:

- 0 - No DR
- 1 - Mild
- 2 - Moderate
- 3 - Severe
- 4 - Proliferative DR

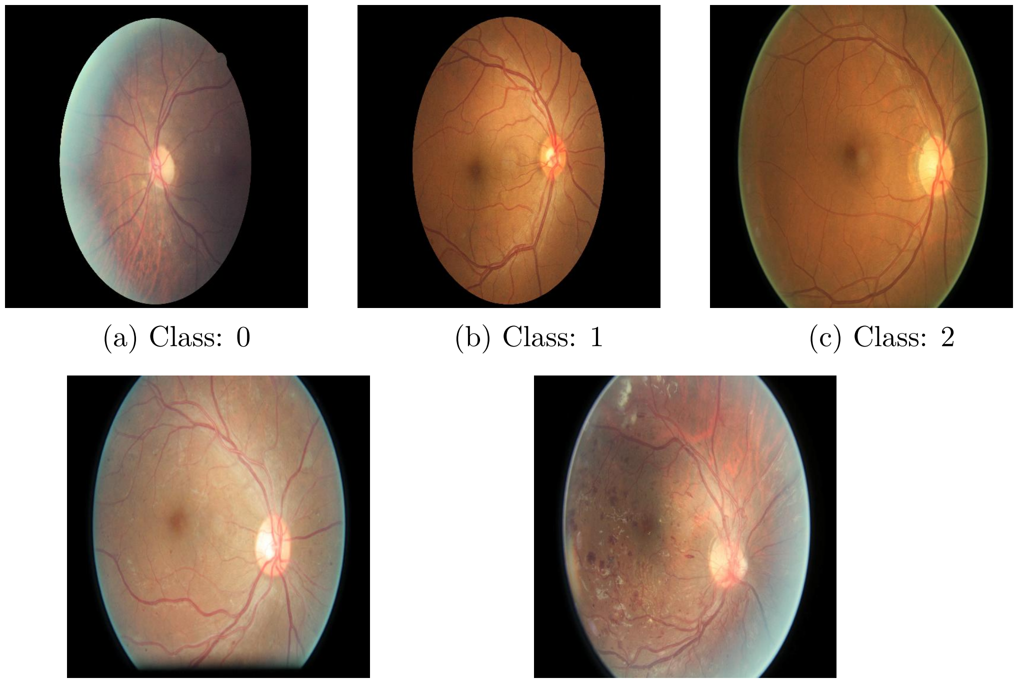
In many papers like Pratt et al. (2016), Wang et al. (2017), Voets et al. (2018) and Quellec et al. (2017) over 88000 images are used. In these papers the dataset is an assemble of training and test data. In this work only the training dataset of the challenge is available which contains 35124 images. One property of this dataset is the imbalance through the classes. As shown in Figure 1 class 0 consists of rounded 73.5% of the data.

Table 1: Counting the number of images per class.

Class	Data	Percentage
0	25808	73.48
1	2443	6.96
2	5292	15.07
3	873	2.49
4	708	2.02

In Figure 1 one example colour fundus image per class is displayed. It is noticeable that images differ in size and brightness.

Figure 1: Example of colour fundus images from the Kaggle dataset; one image per class



(a) Class: 0 (b) Class: 1 (c) Class: 2

(d) Class: 3 (e) Class: 4

Problem

The first step of analysing data is to define the problem. Here the data consists of images which will be used as the variables and the state of DR as the response variable. The state of DR is labelled in five classes (see Chapter 3). So the response variable is discrete which leads to a classification problem. The idea behind classification is to find discriminant functions of hyperplanes that separate the input variables into different classes. Different approaches can be used such as logistic regression, K-nearest neighbors and linear discriminant analysis (Aggarwal, 2018, p. 127). Newer and more complex models are generalized additive models, trees, random forests, boosting and support vector machines (Aggarwal, 2018, p. 127). For image classification the most computer-intensive, but in recent years most successful method is convolutional neural networks. In this thesis neural networks are used to classify images into the five different classes.

As stated above the labels for the state of DR are discrete. In Chapter 3 the classes are explained as going from no DR (Diabetic Retinopathy) to proliferative DR. This means that the classes are ordered. In Fahrmeir et al. (2013) methods like cumulative or sequential models are described to deal with ordinal data. With neural networks ordinal data are usually tackled either as a classification problem where the ordering is not considered or as a regression problem where the distance between classes is not known and it is assumed that labels are real valued. Other ideas exist to deal with ordinal response data for neural networks. Cheng et al. (2008) implemented the so called “NNRank” using a binary cross-entropy or squared error. In Niu et al. (2016) an ordinal regression problem with m ranks/classes is transformed to $m-1$ binary classifier. For each of the $m-1$ binary classifiers it is predicted if the true label is larger than one of the $m-1$ ranks. In Y. Liu et al. (2018) an algorithm is introduced where the loss function is calculated through a composition of the softmax function and a multinomial logistic regression loss. This function is constrained so that the mapped values should be equal or larger than the true label k. In Beckham et al. (2017) an unimodal distribution like the poisson or binomial distribution is stacked on the softmax output so that the distribution over the predicted classes supports classes close to the true label.

A different approach is used in Torre et al. (2018). Here the weighted kappa score is transformed into a loss function. Often the weighted kappa score is used as an accuracy measure to determine the fit of the predicted and the true labels, e.g. in the Kaggle challenge for diabetic retinopathy. In the loss function the discrepancy between predicted and true values is constrained so that it is not of the same weight to the loss function if the difference between predicted and true value is e.g. one or two classes.

Another characteristic of the dataset is the imbalance which has already been displayed in Table 4. There are basically three approaches to tackle this problem. Oversampling is an approach where different techniques are used to increase the available data in smaller classes. Undersampling uses different methods to decrease the number of available data in the larger classes. Finally, the loss function can be weighted to give classes with a smaller number of data more weight in the loss function output. Details are revealed in Chapter 5.

Automated medical image analysis could lead to great improvements in treatments of patients and research (see Chapter 2) and the current most promising method is deep learning. Deep Learning algorithms work the best when the data sets are really big such as the imagenet database (Deng et al. 2009) for the imagenet image recognition challenge (Russakovsky et al. 2015). Despite growing datasets in the context of medicine the datasets are still rather small but the actual questions are complex. So the main question that is investigated in this thesis is, if neural networks are able to maintain good performance on smaller datasets or if other techniques can deliver better results? So the main goal of this thesis is to reduce the size of the training data and observe how this influences the results on the test data.

The diabetic retinopathy datasets is a large image dataset. This has two good properties for the above described problem. The first property is that the dataset is large, so it is possible to find a reliable benchmark score. The second property is that the dataset consists of images. Convolutional neural networks is maybe the field in deep learning where most research is done, so the data falls into a field where the research is not in the very beginnings of research but already deliver applicable methods. The above described problems of imbalance and ordinal regression are also good representations of many medical datasets.

4 Neural Networks

Neural networks are considered as a part of Artificial Intelligence. They are designed as an attempt to simulate the human nervous system, where neurons are connected regions called synapses (Aggarwal, 2018). In statistics, neural networks are used as classification or regression problem solving models. They are known as state-of-the art techniques, particularly in image recognition. The most widely known image classification online challenge is the imagenet challenge (Russakovsky et al. 2015). In this challenge over 14 Mio images from the imagenet database (Deng et al. 2009) have to be classified in 1000 classes. Since 2012 neural networks deliver the best performance in these challenges, even surpassing the human ability to classify these images (Aggarwal, 2018, p. 316). In this chapter especially convolutional neural networks are introduced.

Introduction and Basics

The goal of supervised learning in general and in deep learning is to approximate a function f^* that is used to predict an outcome y using an input x (see e.g. Breiman et al. (2001)). In the case of analysing colour fundus images the images x are used to predict the state of DR y.

Neural Networks contain three different types of layer: input layer, hidden layers and output layer. The input layer takes the input x and distributes it to the hidden layer. The hidden layers process the input through different layers, which can be viewed as the process of learning (O'Shea et al. 2015). The output layer computes a value which should be close or equal to the true value y. The output value can be of different types depending on the statistical problem. Afterwards the difference between the predicted and the true value is calculated using the loss function. Depending on the loss the weights of the layers in the network are adjusted (Goodfellow et al. 2016).

Convolutional Neural Networks

Convolutional neural networks led to major improvements in image recognition tasks, especially between 2011 and 2015 as the top-5 error-rate decreased from over 25% to around 4% (Aggarwal, 2018) in the imagenet challenge (Deng et al. 2009). New architectures as Inception V3 (Szegedy, W. Liu, et al. 2015) and ResNet (He et al. 2016) reach the highest classification accuracies. The idea of convolutional neural networks is based on the visual cortex of a cat. In this visual cortex different objects are causing different regions of cells to be excited or in other words cells are going to be activated based on the shape and orientation of objects. Through a layered connection of the cells the idea of layers providing different portions of images at different layers is assigned from the cats visual cortex to convolutional neural networks. This idea evolved through the years and led to concepts like weight sharing.

In recent years the main change that led to improved results is that more layers and so deeper architectures are used. This is possible because computational power has improved and different techniques are used to regularize and optimize memory and speed requirements through the training process of a neural network (see Chapter 4).

There are a few characteristics of convolutional neural networks such as convolutional layers, the ReLu activation function and pooling layers that differentiate convolutional neural networks from other types of neural networks. Convolutional layers add a specialised handling of grid like data to neural networks which helps in analysing images or time series data. Fully connected layer and linear classifier are introduced as well, but are not typical for convolutional neural networks because they are used in other neural networks as well.

Convolution

In its basic form convolution means that many inputs are provided and transformed into one output via two functions. Mathematically there is a function s providing a smoothed estimate of the input x at time point t:

$$s(t) = (x * w)(t)$$

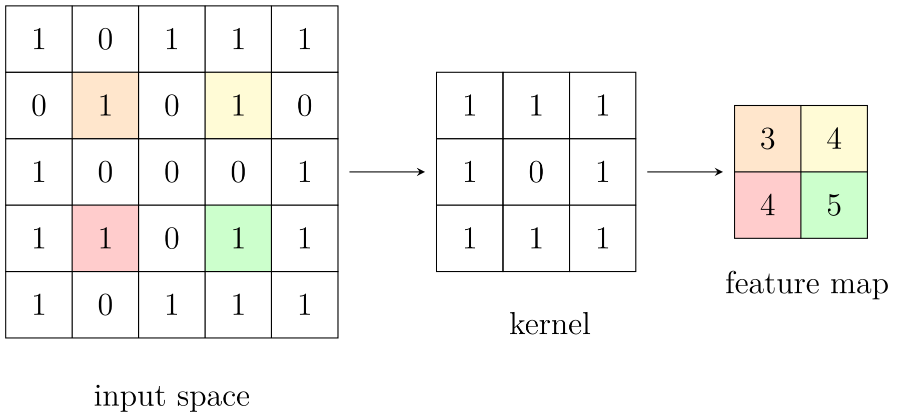
(Goodfellow et al. 2016). Here x is the input data and w is the so called kernel. The dimension of the input space to a convolutional layer can be defined as $n_q \times n_q \times d_q$ for the q-th layer. The kernel, also called filter, in the q-th layer always has the same depth d_q as the input space. The width and height of the input space are usually the same, so that the input space, also called spatial input field, is a square. This is also common for the kernel where the dimension can be defined as $F_q \times F_q \times d_q$ (Aggarwal, 2018). Common values for F are three or five. In the inception network a kernel size of one is also used (see Chapter 4). The kernel is applied to every possible position of the input space. So the number of possible positions defines the width and height of the output space which is the next hidden layer. The width and height of the next layer are defined by

$$n_{q+1} = n_q - F_q + 1. \tag{1}$$

This is the case only when the filter does not stick out at the borders. The depth of the next hidden layer is a manually defined parameter which is defined in the architecture of the network. In other words, one can decide how many filters are applied to the input space and the filter are independent sets of parameters (Aggarwal, 2018). One obvious result of the convolution operation is that the width and height of the next hidden layer are dependent on the number of possible positions of the filter which means that the dimension of the next layer depends mainly on the dimension of the filter. There are two possible parameters that can be set to change the dimension of the next hidden layer. These are stride and padding.

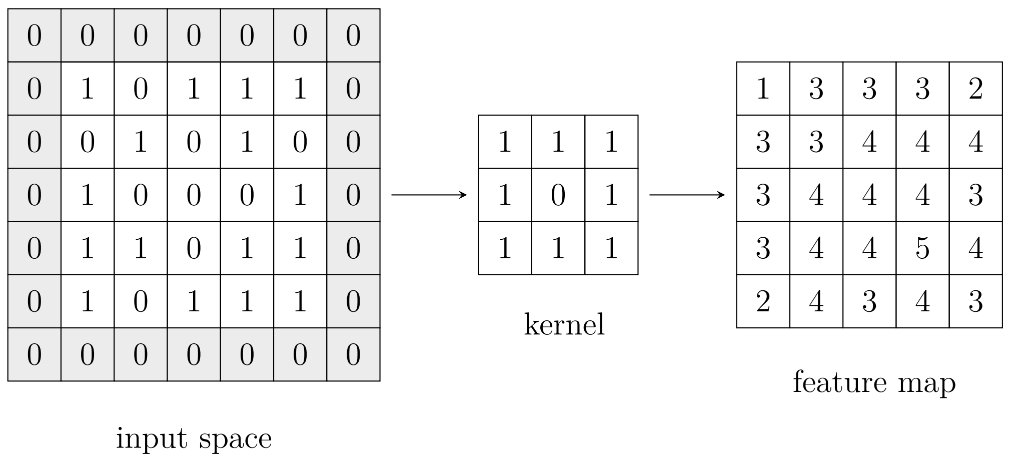
Stride is defined as the jump of the kernel from the current to the next pixel centre. This means that the location of the kernel centre changes by S in both dimensions for an image (Aggarwal, 2018). The higher the stride is the smaller the resulting feature map will be. In Figure 2 a convolution operation is shown with a 3×3 kernel and a stride of 2. The convolution operation would usually result in a feature map of dimension 3×3. The effect of a stride of 2 not 1 is that the dimension of the feature map is reduced to 2×2.

Figure 2: Example of a convolution operation using a 3x3 kernel and a stride of 2 with. The numbers are randomly selected. The kernel centres are coloured to the depending feature map output.



Padding is used when it is not beneficial to reduce the dimension of the feature map because this means a loss of information along the borders of the input space. To increase the dimension of the feature map pixels are added around the input space. Under the assumption that the dimension of the input space and the next hidden layer should be the same, the number of pixels that are added around the input space is defined by $(F_q - 1)/2$ (Aggarwal, 2018). This increases the width and height by $F_q - 1$ and is exactly the loss of dimension due to the convolution operation (see Formula 1). The values of the pixels is 0 which is called *zero padding*. In Figure 3

Figure 3: Example of a convolution operation using a 3x3 kernel with padding zero padding and a stride of 1. The numbers are the same is in Figure 2.



the dimension of the feature map increases to 5×5 if zero padding and a stride of 1 is used compared to Figure 2.

The benefits of the convolution operation are tripartite: sparse interactions, parameter sharing and equivariant representations (Goodfellow et al. 2016). In traditional networks every input neuron is connected to every output neuron (see Chapter 4). Using a kernel for convolution reduces the number of weights to the kernel size. This also results in parameter sharing. As above mentioned traditionally every input neuron is connected to every output neuron. This means that each weight is only used for one connection. The convolution operation uses every kernel location on every input location except for boundary cases and a stride higher than one. So instead of using weights for every connection between input and output, the number of weights is limited to the size of the kernel. This reduces the storage requirements heavily. The last benefit is equivariance to translation. Equivariance means that the output changes in the same way as the input. In Goodfellow et al. (2016, p. 334) this property is defined as follows:

$$\text{A function } f(x) \text{ is equivariance to a function } g \text{ if}$$
$$f(g(x)) = g(f(x)).$$

Activation

Activation functions decide on whether a neuron will be activated, which means that the output of the activation function states whether the output of the neuron gives relevant information. An activation function is non-linear, monotonically increasing and bounded. Typical activation functions are the sigmoid function, the hyperbolic tangent and the ReLU (Rectified Linear Unit) function. In most applications the ReLU function is used which is defined as

$$\phi(\nu) = max\{\nu, 0\}$$

(Aggarwal, 2018). The ReLU function converts negative values to zero. This means that not all neurons are activated which results in efficient and easy computation. Also the gradients can be zero, so that the weights are not updated. Downside of this is that this sometimes leads to dead neurons, meaning that these weights are never updated.

Pooling

Another typical part of convolutional networks are pooling layers. The pooling function summarises a neighbourhood of the dimension $n \times n$ into one value. It is supposed to make outputs invariant to translations in the input space, so that smaller changes in the input space do not change the output (Goodfellow et al. 2016). The goal is also to cancel out less crucial information. For that a filter with $n \times n$ dimension is defined and a stride similar to the stride used in the convolution operation. Possible pooling methods are maximum, average and global pooling. Max-pooling is used most of the times and takes the maximum value of a $n \times n$ region. Pooling is commonly used with a stride larger than 1 so that the dimension of each activation map is reduced. Max-pooling is defined by

$$a_j = \max_{N \times N}(a_i^{n \times n} \mu(n, n)),$$

where a_i are the pixel values and $\mu(n, n)$ defines the window function with the dimension $n \times n$ (Scherer et al. 2010). Global pooling means that the pooling filter has the same size as the input layer. If average pooling is used the average of the defined neighbourhood space is taken (Aggarwal, 2018).

Fully Connected

A fully connected layer is functioning in the same way as a feed-forward neural networks. In fully connected layers all neurons are connected to the neurons in the previous layer. In convolutional neural networks this is used after the last spatial layer. This is of advantage, especially at the end of convolutional networks because there is much more power in the computations. The problem is that fully connected layers need a lot parameters. E.g. if there are 4096 hidden units in two successive fully connected layers, then there are 16777216 million weights between these two layers (Aggarwal, 2018).

Linear Classifier

The linear classifier is the last layer of a neural network and so the output layer of the network. It computes the output values for each class in the case of a classification problem. The softmax activation function is the most common used function in multinomial logistic regression or classification problems because it can be seen as a generalization of the sigmoid function which is used for binary variables (Goodfellow et al. 2016). The softmax function is defined through

$$softmax(z)_i = \frac{exp(z_i)}{\sum_j exp(z_j)}.$$

The value z is the logarithmic probability that $y = i$ given the input x:

$$z_i = logP(y = i|x).$$

This can be written to be consistent with the form of a neural network:

$$z = W^\top h + b,$$

where W are the weights of the last hidden layer with the neurons h and b is the bias of this layer (Goodfellow et al. 2016, p. 181). In all cases $i, j = 1, \ldots, n$ and n defines the number of classes. So the softmax layer is exponentiating and normalizing the output of the last layer.

Architecture / Inception V3

From basic feed-forward neural networks to recurrent neural networks - network architectures are manifold and have changed and improved over the years. In image analysis convolutional neural networks are proven to be better suited. In the imagenet challenge (Russakovsky et al. 2015) new architectures are developed to improve prediction accuracy of image classification. In 2012 the so called "alexnet" won the competition with a top-5 error of 16%. In 2017 the best result is at 2.3%. The most famous and known architectures are VGG (Simonyan et al. 2014), Resnet (He et al. 2016) and Inception models (Szegedy, W. Liu, et al. 2015) delivering results of 23.7%, 19.38% and 17.3% top-1 error. So the current top-1 error is almost as good as the top-5 error from 2012 which shows the fast improvement in the field of deep learning and image recognition.

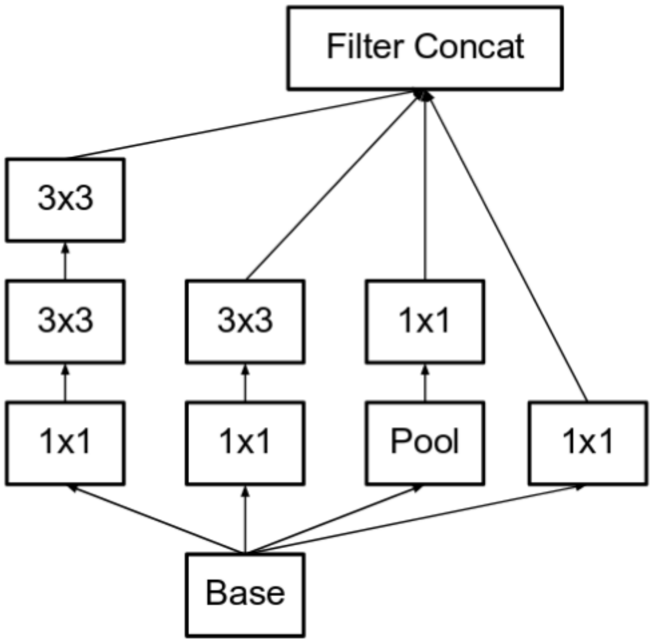
Inception Layers

One evolution has been to just stack more and more of convolutional layers with different kernel sizes. This has two disadvantages: It is prone to overfitting and computationally expensive. From images the parts with valuable information differ also in size and position. So the choice of the right kernel size in convolutional layers is important to find different valuable information either on a global or a local level. The inception network has been introduced in Szegedy, W. Liu, et al. (2015) and revised in Szegedy, Vanhoucke, et al. (2016). An essential part of the network architecture is the inception layer which tries to tackle the problem of just deepening networks and finding different patterns in images.

The idea of the inception layer is to compute multiple convolution operations with kernel sizes 1×1, 3×3 and 5×5 in parallel and concatenate them afterwards (Szegedy, W. Liu, et al. 2015). As 1×1 convolution operations are computationally much cheaper because of less parameter, these computations are performed additionally before the 3×3 and 5×5 layers. Parallel to these operations a max-pooling layer following a 1×1 convolution is performed.

In Szegedy, Vanhoucke, et al. (2016) the inception layers are further revised. The 5×5 convolution is replaced by two 3×3 convolutions (see Figure 4). The reason is that the operation of one 5×5 convolution compared to one 3×3 convolution is $25/9 = 2.78$ computationally more expensive. The disadvantage is that less activation signals that are further away from each other are captured. In multiple experiments the authors state that this change in the architecture brought improvements in terms of validation accuracy.

Figure 4: Inception layer where 5×5 convolution is replaced by two 3×3 convolutions. Abstracted from Szegedy, Vanhoucke, et al. (2016)



Because a reduction of the dimension of the kernel sizes is suggested above, the question is if further reducing the dimension leads to even more improvements. The authors of Szegedy, Vanhoucke, et al. 2016 found out that a change from $n \times n$ convolutions to asymmetric $1 \times n$ and an $n \times 1$ convolution (see Figure 5) is leading to the same spatial fields. For $n = 3$ the savings in terms of computational resources is 33%. In practice this works best on medium grid-sizes. Best results are achieved with $n = 7$.

A variation of this can be found in Figure 6. Here one 3×3 convolution stays and the other ones are replaced by 1×3 and 3×1 convolutions which are used in parallel and not successively as in Figure 5. This results in three different inception layers in Figures 4, 5 and 6.

Computational Graph

Although the inception architecture tries to avoid going just deeper using the inception layers it is still a very deep network prone to overfitting and the vanishing gradient problem (see Chapter 4). In the inception network two auxiliary classifiers are used with softmax outputs and an auxiliary loss is computed over these. In theory, this should regularize the vanishing gradient problem and help in stable learning and better convergence (Szegedy, Vanhoucke, et al. 2016). But Szegedy, Vanhoucke, et al. (2016) found that auxiliary classifiers mostly help in convergence

Figure 5: Inception layer where 3×3 convolutions are replaced with $1 \times n$ and $n \times 1$ convolutions. Abstracted from Szegedy, Vanhoucke, et al. (2016)

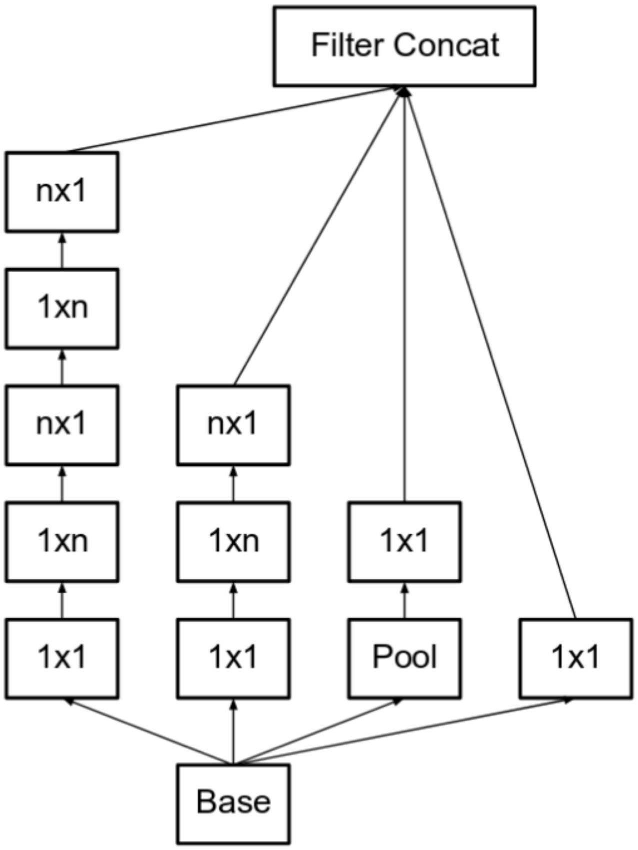
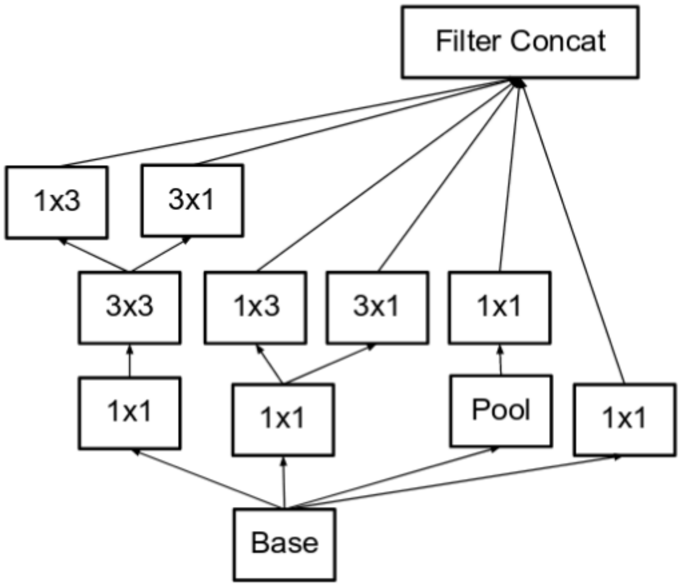


Figure 6: Inception layer where 3×3 as well as 1×3 and 3×1 convolutions are used. Abstracted from Szegedy, Vanhoucke, et al. (2016)



at the end of training and do not change the weights in the beginning. The total loss is a weighted sum of the real loss at the end and the auxiliary losses weighted with the factor 0.3. In version three of the inception network only one auxiliary classifier is included. This layer is placed after the 7th Inception layer.

The inception v3 networks architecture is displayed in Table 2.

Table 2: The architecture of the inception v3 network, extracted from Szegedy, Vanhoucke, et al. (2016).

type	patch size / stride	input size
conv	3×3 / 2	299×299×3
conv	3 ×3 / 1	149 × 149 × 32
conv padded	3 × 3 / 1 147 × 147 × 32	
pool	3 × 3 / 2	147 × 147 × 64
conv	3 × 3 / 1	73 × 73 × 64
conv	3 × 3 / 2	71 × 71 × 80
conv	3 × 3 / 1	35 × 35 × 192
3 × Inception	As in Figure 4	35 × 35 × 288
5 × Inception	As in Figure 5	17 × 17 × 768
2 × Inception	As in Figure 6	8 × 8 × 1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Training

Training a neural network consists of two parts: Forward- and back-propagation. In the forward-propagation part the inputs are fed through the network and the output is calculated. Afterwards a loss function is applied to find the difference between predicted and true values. This loss function differs amongst applications, e.g. between classification and regression problems. In the back-propagation process the partial derivatives are calculated in each layer. Through stochastic gradient descent (Chapter 4) the weights and biases are updated.

The weights are the connections between the nodes of the layer. These can either be the weights or the value of kernels as for the convolution operation or the weights of the fully connected layer. In all cases it is possible to add a bias term to a layer. This bias has the same influence as in the linear regression. The bias is shifting the function across the weights by a given value. This value is also optimized in the training process.

Stochastic Gradient Descent

Stochastic gradient descent (SGD) is the most used optimization algorithm in machine and deep learning. It is based on the concept of gradient descent: The aim of training a neural network is to minimize the loss function, the aggregated difference between predicted and true values. The derivative of a function $f(x)$ gives the slope of $f(x)$ at the point x and shows the direction and angle of the steepest ascent. To make small improvements in y this is helpful to decide on how to change x. Reducing $f(x)$ works through moving x in the direction of the derivative using the opposite sign to go in the direction of descent and not ascent (Goodfellow et al. 2016, p. 80-81). In the case of deep learning x are the weights and y is the output of the loss function.

The difference between gradient descent and stochastic gradient descent is that in in the latter only mini-batches and not the full dataset is used. Mini-batches are subsets of the full training data. As defined in Formula (2) the weights are updated using the gradient of the loss function. This gradient is an expectation and can be estimated by mini-batches (Goodfellow et al. 2016, p. 149). This reduces computation time heavily.

An important parameter in stochastic gradient descent is the learning rate. The learning rate specifies the step size defining how long a jump into the direction of the steepest descent is made. The main purpose of stochastic gradient descent is to adapt the weights of a network, which is defined through the following formula:

$$w_{i+1} = w_i - \alpha \frac{\delta L(y, f(x))}{\delta w}, \tag{2}$$

where w_i are the weights in epoch i, α is the learning rate and $L(\cdot, \cdot)$ is the loss function.

Back-propagation

Back-propagation has been introduced in 1986 by Rumelhart et al. (1986). Weights and biases explain the contribution of each pixel to the loss function respectively the predicted values. In order to minimize the loss function and so to maximize the accuracy of the neural network weights and biases have to be changed. In the back-propagation process the error contribution of each weight is calculated. Important here is to differentiate that the back-propagation process describes the method of computing the gradients and not the weight update, for which e.g. stochastic gradient descent is used (Goodfellow et al. 2016, p. 200). The difficulty is that the neural network consists of many different layers with a lot of different weights. For all layers the derivatives or the contribution of the weights is changing. To compute the gradients for the weights in different layers, dynamic programming is used, which is the chain rule of differential calculus (see e.g. Goodfellow et al. 2016, p. 201). With the chain rule derivatives of the composition of multiple functions, where the derivative is known, are computed. In the univariate approach of the chain rule the derivative of a function is not computed with respect to a recursively derived argument but with respect to its immediate argument. With this technique the derivatives of all layers can be computed directly to its immediate argument. The gradient of the loss function with respect to the weights that should be updated is the product of the local gradients along the so called computational graph (see Goodfellow et al. (2016, p. 201)). An example is also given in Goodfellow et al. (2016, p. 201). If two functions $y = g(x)$ and $z = f(g(x)) = f(y)$ are given, then the chain rule results in

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

Here the derivatives $\frac{dz}{dy}$ and $\frac{dy}{dx}$ are derived to its immediate argument. The composition of these both derivatives results in the derivative of the function z to the weights x that are needed. This can also be written in a vectorised notation:

$$\nabla_x z = \left(\frac{\delta y}{\delta x}\right)^{\top} \nabla_y z$$

(Goodfellow et al. 2016).

Loss Function (Objective Function)

The loss function measures the distance between predicted and true values. In regression problems a typical loss function is the Mean Squared Error (MSE) or L2 loss. The MSE is defined as

$$L(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2$$

where y is the true value and $\hat{y}$ is the predicted value (James et al. 2013, p. 29). This calculates the average of the squared distances.

For classification Cross-Entropy loss is mostly used and defined as:

$$L(y_i,\hat{y}_i) = -\sum_{c=1}^{M} y_{i,c} log(f_k(p_I, c))$$

where $y_{i,c}$ defines if the class label c is correct for the i-th observation, $p_{i,c}$ is the predicted probability for class c of the i-th observation and M is the number of classes (James et al. 2013).

Optimization and Regularization

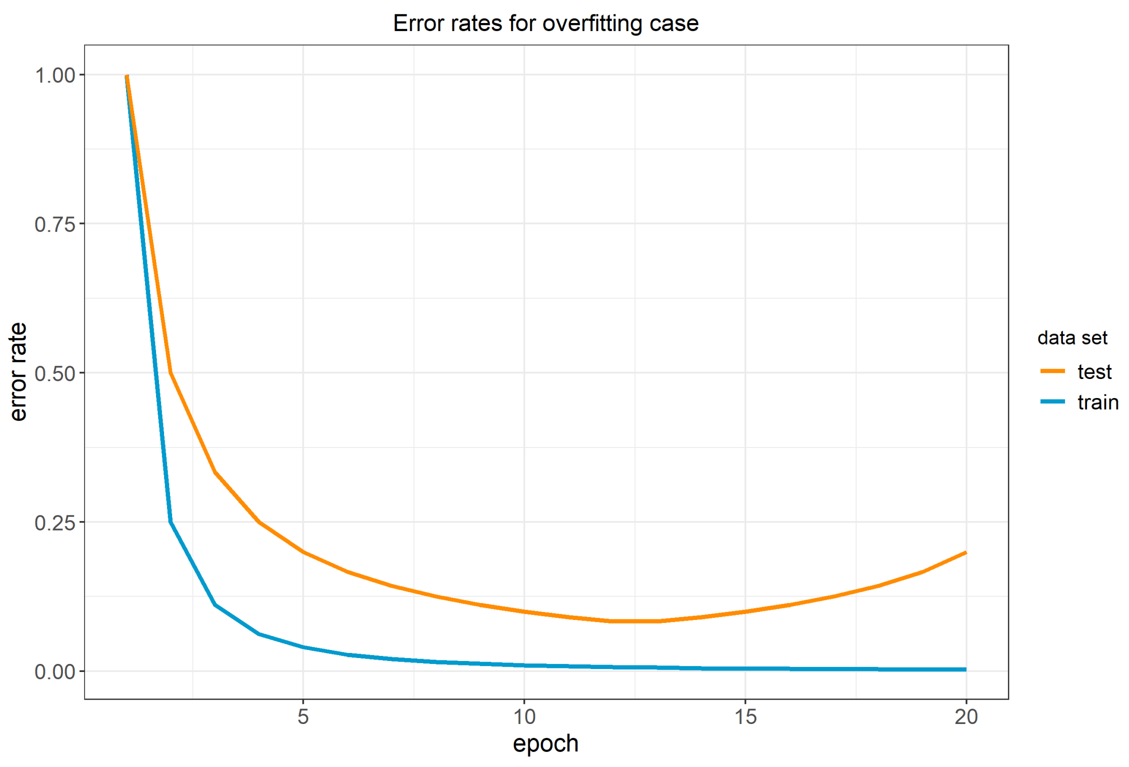
Training neural networks is a complex process in which multiple issues can arise (Aggarwal, 2018):

- Overfitting
- Vanishing and Exploding Gradients
- Convergence difficulties
- Local Optima

Overfitting is the most common problem and arises in almost any machine learning problem. While the model error rate is decreasing for the training data, it can be the case that the error rate on the unseen test data is already increasing again. This behaviour can be seen in Figure 7. This means that the model cannot be generalized to unseen data. The model then learns patterns in the training data that do not generalize to the test data and are specific for the training data.

Vanishing and exploding gradients mean that in earlier layers the gradients can be really small or really high. In the case of vanishing gradients this can prevent the weights from changing at all. If the gradients explode the opposite is the case and

Figure 7: Example for training and test error rates in the case of overfitting. Values are self created.



the weight update is really high. In both cases, this means that the neural network is unstable. This is slowing down the training process and convergence issues can arise. This occurs especially in very deep neural networks with many layers and is caused by the chain-rule product computation (see Chapter 4).

Difficulties in convergence means that the loss function or accuracy does not convert to the best results.

Convergence means also that the goal is to reach a global optimum and not a local one. In more complex functions a higher amount of local optima exists which leads to a higher probability to get stuck in these. As neural networks tend to be very complex a challenge is to avoid local minima and find the global minimum.

Following are some principles and methods presented which help to overcome these issues.

Batch Normalization

As mentioned earlier mini-batches of the training data are used in forward- and back-propagation to optimize the weights of a network. Batch Normalization, introduced in Ioffe et al. (2015), is applied as a layer that is included in the network architecture. In this layer the input is normalized so that the mini-batch has a fixed mean and a fixed variance in every training instance. Usually batch normalization is used either just after an activation function (post-activation values) or just before the activation function (pre-activation function) (Aggarwal, 2018). The batch normalization is done with following formulas (Ioffe et al. 2015):

$$\begin{aligned}
\mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=1} m x_i && \text{(mini-batch mean)}\\
\sigma_{\mathcal{B}}^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_{\mathcal{B}})^2 && \text{(mini-batch variance)}\\
\hat{x}_i &= \frac{x_i-\mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}+\epsilon}} && \text{(normalize)}\\
y_i &= \gamma\hat{x}_i+\beta = \mathrm{BN}_{\gamma,\beta}(x_i) && \text{(scale and shift)},
\end{aligned}$$

where x is the input of a mini-batch $\mathcal{B}=\{x_{1\ldots m}\}$, m is the size of the mini-batch and γ, β are the parameters that are learned. The constant ϵ is added to reach numerical stability.

Batch normalization is especially known to be helpful for vanishing and exploding gradients. The distribution of layer inputs changes continuously throughout the network architecture, also called internal covariate shift (Aggarwal, 2018, p. 152).

Obviously small changes in earlier layers can lead to big changes in later layers. Through batch normalization the layer inputs are kept stable. Other outcomes of batch normalization are that a higher learning rate can still lead to convergence, escaping of local minima can be reached and that there is a regularization effect (Aggarwal, 2018).

Recent research in (Santurkar et al. 2018) shows that the real effect of Batch Normalization is not about the internal covariate shift, which is widely concluded, but about smoothing the landscape of the optimization problem significantly. The possible larger range of learning rates and faster convergence of the network are the real advantages of Batch Normalization.

Momentum

Momentum is a technique to accelerate the learning process in cases of high curvature, small but consistent or noisy gradients. In the weight update rule a new parameter ν is introduced as velocity:

$$\begin{aligned} \nu &\leftarrow \epsilon\nu - \alpha\nabla_{w_i}\left(\frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};w_i),y^{(i)})\right), \\ w_{i+1} &\leftarrow w_i + \nu, \end{aligned}$$

where α is the learning rate and $L()$ is the loss function of the true value y and the predicted value $f(x^{(i)};w_i)$ (Goodfellow et al. 2016, p. 293). The parameter m is defining the number of past gradients that are taken into account. ϵ is the momentum parameter which defines the contribution of previous gradients, particularly the velocity of the exponential decay. The ratio between ϵ and α defines the number of gradients that affect the current direction of the descending step. All this means that the new parameter ν is set to an exponentially decaying average of the negative gradient so that past gradients are accumulated. As the learning rate is the step size, this means that if multiple successive gradients point in the same direction the learning rate is larger.

Adaptive Learning Rate

Momentum should accelerate the learning process by avoiding zigzagging for partial derivatives while changing or adapting the learning rate. The idea behind adaptive learning rates is to adapt the learning rate for every parameter (Aggarwal, 2018, Chapter 3.5.3). Over the years many approaches were introduced. The most common one is Adam which is used in this thesis and therefore only Adam is explained.

The advantages of Adam are that the method uses only first-order gradients and do only need small memory requirements. The method is a combination of the advantages of the AdaGrad and RMSProp methods (Kingma et al. 2014). Adam uses the first and second moments of the partial gradients and similar to the momentum method keeps past gradients. This is done with an exponentially decaying average for first and second moments.

In a formal way: The exponentially averaged value A_i of the i-th parameter w_i is updated with the decay parameter $\rho \in (0, 1)$ using the second-order gradient (or second moment):

$$A_i \leftarrow \rho A_i + (1-\rho)\left(\frac{\delta L}{\delta w_i}\right)^2 \quad \forall i$$

(Aggarwal, 2018, Chapter 3.5.3.5). Also an exponentially averaged value of the gradient F_i of the i-th component is smoothed with a different decay parameter ρ_f using the first moment:

$$F_i \leftarrow \rho_f F_i + (1-\rho_f)\left(\frac{\delta L}{\delta w_i}\right) \quad \forall i$$

The parameter w_i is updated using the predefined learning rate α_t and the updated parameters A_i and F_i:

$$w_i \leftarrow w_i - \frac{\alpha_t}{\sqrt{A_i}} F_i \quad \forall i.$$

The decay rate is to be set by the user. E.g. the default values in the pytorch framework (Paszke et al. 2017) are 0.9 or 0.999.

In Reddi et al. (2018) a new varied version of the Adam algorithm is introduced using “long-term memory” of past gradients which should avoid convergence issues in cases with large output spaces. This should also lead to improvements in empirical performance.

An alternative to new adaptive learning rate algorithms like Adam or AdaGrad is e.g. momentum stochastic gradient descent. In (Zhang et al. 2017) adaptive learning rates are compared with momentum stochastic gradient descent. After finding the competitiveness of the latter robustness is measured and an new automatic hyperparameter tuner called YellwoFin is introduced. YellowFin is tuning momentum and learning rate parameter simultaneously.

Early Stopping

As discussed before in every machine learning model overfitting is a problem. This means that the training error is still decreasing but the validation error is either steady or even increasing. One procedure to prevent overfitting in deep learning is early stopping. In early stopping the algorithm terminates if for a specified number of epochs the validation set error does not decrease (Goodfellow et al. 2016, Chapter 7.8). Important here is that the validation dataset, is part of the training dataset but is never seen during the training period. If this is the case the algorithm returns the parameters of the epoch with the lowest validation set error and not of the latest epoch.

Dataset Augmentation

One part of overfitting is that the algorithm does not generalize to unseen data. A possible reason is that the training dataset is too small. As in most cases it is hard to get more “real” data. Augmenting data can solve this problem. Data augmentation means that the data points are transformed in many possible ways without changing the label of the data (Goodfellow et al. 2016, Chapter 7.4). An example for a change of the label is if a “6” is vertically flipped, this would result in a “9”. Typical data augmentation techniques are rotations, flipping or scaling. New approaches as skewing, distortion or hist equalization are effective techniques as well.

Rotations: Images are being rotated by a pre-defined angle. Either all images can be rotated by the same fixed angle or all images are rotated randomly. Here a minimum and maximum angle is pre-defined. Using the Augmentor framework (Bloice et al. 2017) for data augmentation it is also possible to define if the corners should be cropped when images are rotated.

Flips: Flips mirror the image either along the horizontal or the vertical axis.

Scaling: The size of the image is either increased or decreased by a defining factor. This does not lead to zooming into an image, but to increase the number of pixels.

Skewing: Corner or Sides of images are skewed into the background.

Distortion: A defined grid is distorted. This means parts of the images are either brought into the back- or foreground.

Hist Equalization: This is a contrast enhancement technique. The colour distribution of images can be displayed by a histogram. Here the number of pixels is counted for each value. The aim of Hist Equalization is a uniform histogram. This results in larger contrasts in the image, which should emphasize the differences between images of different classes (Makandar et al. 2014).

ColorJitter: The image is changed in brightness, contrast and saturation.

In Deep Learning two different techniques exist to apply data augmentation. These are called offline and online data augmentation. Offline data augmentation applies the augmentation technique on the image and creates an actual new image. These images have to be stored on disk. When online data augmentation, also called realtime data augmentation, is the chosen method, then images are going to be augmented during the training. The advantage is that the images are not going to be saved on disk, but less control over the augmentation process is possible.

L2 Regularization of loss

In classical regression problems L2 regularization is e.g. known as ridge regression. The principal can be easily transferred to deep learning problems. As in ridge regression a penalty term is added to the loss function. This penalty is called regularizer and in the case of weight decay it is defined as

$$\Omega(w) = w^\top w$$

(Goodfellow et al. 2016, Chapter 5.2). A parameter λ can be used to control the strength of the penalty term. When using the Mean Squared Error loss a penalized loss function would look like this:

$$L(y, \hat{y}) = MSE_{train} + \lambda w^\top w.$$

Dropout

Dropout can be understood as an ensemble method like bagging. In bagging multiple models are trained and evaluated. The models are usually unbiased, but have a high variance. Training multiple different neural networks and ensemble them is computationally really expensive. In the dropout technique neuron or nodes in the input and hidden layer are randomly dropped using predefined probabilities. So in different epochs different parameters are trained and shared (Goodfellow et al. 2016, Chapter 7.12). The probability to drop an input node is usually 20% and to drop an hidden node is 50%. The probabilities and which node is going to be dropped is independent for every layer (Aggarwal, 2018, Chapter 4.5.4). In the training process of a neural network different parameters are trained. These parameters are shared so that dropout is node sampling with weight sharing. This sampling process is applied on every mini-batch training step.

When evaluating a bagging model the test dataset is applied on every model and the results are aggregated. In neural networks this process is not necessary as forward-propagation is performed using the base network without dropping nodes. The approach for this is the weight scaling inference rule where the weights are multiplied by the probability of including unit i. So if a weight is dropped with a probability of 50% the weight is divided by 2 in the test set prediction phase (Goodfellow et al. 2016, p. 260). Compared to other ensemble methods this means that every epoch is an own model and these models are assembled after every epoch through the weight scaling inference rule.

Mixed Precision Training

Year by Year neural networks are trained on larger datasets and deeper network architectures. In Micikevicius et al. (2017) mixed precision training is introduced as a technique to reduce memory requirements and to speed up training on multiple GPUs. Usually single-precision training (FP32) is used to train neural networks. Mixed precision training uses half precision training (FP16). Half precision training means that half the number of bits in the floating point format (so 16-bits instead of 32 bits) is used. A downside of this technique is that really small weights often become zeros because of limited representation of numbers. Three techniques are used to still match the model accuracy of FP32 trainings.

1. FP32 Master Copy of Weights
2. Loss Scaling
3. Arithmetic Precision

The reason for a master copy of weights in FP32 precision is the above explained behaviour of really small weights that become zero. The weight update is done in half precision and then weights are saved in single precision. Loss Scaling has the same reason as gradients become zero because the gradients are too low to be represented larger than zero by the limited number of bits. Through scaling up the gradients a much higher percentage of the gradients become representable. So a lot lower number of weights become zero. Here either a static scaling factor can be defined or a dynamic loss factor can be used. A really high value for the scaling factor can lead to an overflow of gradients which means that gradients become infinite of NaNs. Dynamic scaling controls this problem with the procedure that if an overflow occurs the weight update is skipped and the scaling factor is reduced. If no overflow occurs in multiple iterations the scaling factor is increased again. A scaling factor near the maximum value that causes no overflow is preferable as a larger number of gradients does not become zero. The third technique is arithmetic precision. Some arithmetic operations need to be converted from FP16 to FP32 e.g. vector dot-products, large reductions and point-wise operations. For the NVIDIA Volta GPUs the special technique of arithmetic precision is introduced NVIDIA (2017) to eliminate model accuracy loss because operations are done in half precision and not in single precision.

Transfer Learning

Transfer learning is useful in low-volume datasets. If there are two different datasets where the input is the same, which is e.g. an image, but one dataset has significantly more data, transfer learning is often applied. The larger dataset is used to train a neural network from scratch. Then the last fully connected layer is adjusted to the new dataset. The dimension of the last layer is dependent on the number of classes which most of the times differ in two different classification problems. The neural network is now trained on the smaller dataset using the pre-trained weights (Goodfellow et al. 2016, Chapter 15.2).

It is possible to download already pre-trained networks. Usually all frameworks for deep learning have a model zoo. The architectures of pre-trained networks are variations of alexnet, vgg, inception, densenet, resnet and more. For this thesis the networks are pre-trained on the imagenet database (Deng et al. 2009, Imagenet Database) which contains more than 14 Mio images of 1000 classes.

Generally in literature, there is a differentiation between two different techniques: Transfer Learning and Fine Tuning. In Transfer Learning only the last layer is changed and retrained. This is useful for small datasets and guarantees a faster training time. Fine Tuning describes the process of using a pre-trained neural network as an initializer for the weights. This means that all weights are retrained but it is expected that the pre-trained weights are a good base, so that the training process is way shorter compared to training a network from scratch. When a network is fine tuned, the last layer still has to be replaced.

One interesting variation with big influence is the number of layers which are retrained. Here not much literature exists for which setting is the most reasonable. In the paper (Tajbakhsh et al. 2016) a comparison between Full Training and Fine Tuning is made with the results that pre-trained convolutional neural networks (CNNs) outperform networks and that fine-tuned CNNs are more robust to changes in the training set size. Also, different fine tuning techniques as shallow and deep tuning are compared with the results that neither of them, but layer-wise fine tuning could lead to better results.

Evaluation Metrics

The performance of a model on unseen data can be evaluated with multiple different metrics. Generally the goodness-of-fit is measured through the distance between the true and predicted label. Most measures for classification problems are based on the confusion matrix. In Table 3 the confusion matrix is displayed for binary classification problems (Fawcett, 2006).

Table 3: Confusion Matrix for two classes

		Predicted	
		Positive	Negative
True	Positive	True Positives (TP)	False Positives (FP)
	Negative	False Negatives (FN)	True Negatives (TN)

Accuracy, sensitivity and specificity are widely used measures which are based on the confusion matrix. The measures are defined as follows (Fawcett, 2006):

$$\text{Acc} = \frac{TP + TN}{n} \tag{3}$$

$$\text{Sensitivity} = \frac{TP}{P} \tag{4}$$

$$\text{Specificity} = \frac{TN}{FP + TN}, \tag{5}$$

where n is the number of observations. The accuracy defines the ratio between values which are predicted in the correct class and all values. The sensitivity explains the ratio between correct predicted values and true positive values. The specificity explains the ratio between correct classified negative values and all truly negative values. For a classification with five classes sensitivity and specificity cannot be used in the same manner. Here only the sensitivity is used as the ratio of correct classified data points per class, which is the accuracy per class.

In Chapter 3 two main problems of the dataset are discussed. One is the imbalance of the data and the other one is the ordering of the classes. The Cohen's kappa score is used in the Kaggle challenge for diabetic retinopathy Kaggle, 2018, because it covers both above cases. It was introduced in 1960 by Cohen and later advanced to the quadratic weighted kappa score in Cohen (1968):

$$\kappa = \frac{p_0 - p_e}{1 - p_e},$$

where p_0 is the probability of agreement and p_e is the expected agreement when labels are assigned randomly. The weighted kappa score is defined by

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}},$$

where $i, j \in 0, \ldots, 4$ define the classes, $O_{i,j}$ is the number of observations classified in the ith category if the true class is j and $E_{i,j}$ is the outer product product between the two histogram vectors of predicted and true values (Torre et al. 2018). E is normalized so that E and O have the same sum. The weight penalization $w_{i,j}$ is defined by $w_{i,j} = \frac{(i-j)^n}{(C-1)^n}$, where C is the number of classes. The value $n = 1$ leads to linear and $n = 2$ to quadratic penalization. The values of this score are in the interval of $\kappa \in [-1, 1]$, where -1 means perfect symmetric disagreement and 1 perfect agreement.

Matthews correlation coefficient is also builds on the confusion matrix and preferable for imbalanced dataset. It is introduced in (Matthews, 1975) and is defined through the following formula for 2 classes:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}$$

(Chicco, 2017). This can be extended to multiclass problems.

Top 5 accuracy is used in the imagenet challenge (Russakovsky et al. 2015) to see if the true label is in the first five predicted classes which are ordered by predicted probability. Reasons to use a top 5 accuracy is incorrect labeling, incomplete annotations or multiple classes on one image. Generally it can be called top k accuracy. If the true label is in the top k most likely predicted labels, then this counts as a correct predicted image (Berrada et al. 2018). The parameter k should be small compared to the number of classes. In the case of diabetic retinopathy with 5 classes $k = 2$ is used. This measure is used because classes cannot be differentiated very easily and the labels are not considered to be fully reliable (Lam et al. 2018).

5 Proceeding

In this chapter data and theory is combined. All settings are presented that are needed to deal with possible problems that are occurring in training the convolutional neural networks.

Technical Informations

The convolutional neural networks are trained using Python (Rossum, 1995) in combination with the deep learning framework Pytorch (Paszke et al. 2017).

As Deep Learning is computationally expensive and as a lot of runs with different settings and especially different training set sizes is needed, it was not possible to train the neural networks on a local CPU. For that the deep learning system DGX-1 with Tesla V100 of the lrz (*LRZ* 2018) is used which technical details are:

- Peak Performance: 960 TFlop/s
- 8 V100 GPUs
- 128 GB Ram
- 40960 CUDA Cores
- 5120 Tensor Cores

More details are available on the nvidia webpage (nvidia, 2019).

In python standard libraries as pandas (McKinney, 2010) and numpy out of the SciPy ecosystem (Jones et al. 2001) support the framework system. The visualization is done in R (R Core Team, 2018), mainly using the ggplot2 (Wickham, 2016) and xtable (Dahl et al. 2018).

Training, Validation and Test Data

The dataset is introduced in Chapter 3. This dataset is split into three datasets: Training, validation and test set. As earlier introduced this dataset has 5 classes which stands for the state of DR. In these 5 classes the dataset is very imbalanced (see Table 1). Because of this imbalance in the dataset the dataset is split with respect to the classes. First the data is split into training and test dataset with the ratio of 3 to 1, so that 25% of the data are in the test set. Because the number of training data is successively reduced, a subset of the training data is used based on the selected number of images. In the first approach the subset of training data is as balanced as possible through the classes. This means that a number of images per class is calculated based on the overall number of images that is selected for the training set. Afterwards, as many images as possible are selected per class. If in one class less images than needed are available, then this void is filled with images of other classes. In the end the sum of the number of images per class should be equal to the selected overall number of images for the training set. The settings for the exact numbers are discussed in the “settings” chapter later. The subset of training data is then split into training and validation data. 1/6 of the training data is used as the validation data. This split is also done with respect to the class sizes.

In Table 4 and Figure 8 the distribution of the data across training, validation and test data within the different classes can be seen. All available data is used and the imbalance of the dataset can be seen clearly. To avoid that the imbalance of the data has a high influence on the results, several techniques exist. In the paper More (2016) four methods are compared: Weighted loss function, Undersampling, NearMiss-3 and Oversampling. Also, some combinational and ensemble methods are introduced. Best performing are ensemble and combinational methods. The weighted loss function method works really well, particularly considering how easy the implementation of a weighted loss function in pytorch is. In Buda et al. (2018) the imbalance problem is studied for neural networks. Here oversampling achieves the best results.

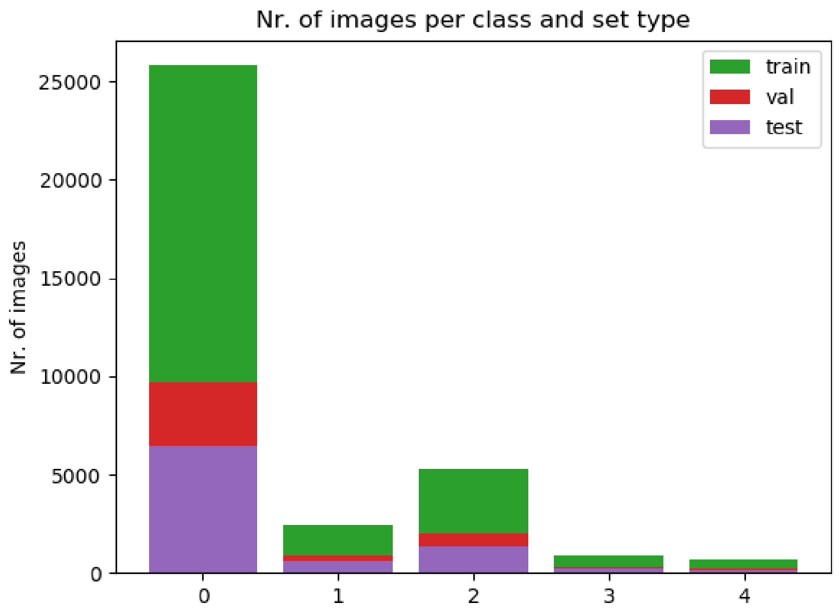
In this thesis the dataset is used with different numbers of images. The problem of imbalanced data only occurs for larger training sets. If this is the case the loss function is going to be weighted. This is preferable mainly because of the shorter computation time. The case of smaller training sets is comparable with undersampling techniques because a subset of data is used and this is balanced throughout the classes.

In Chapter 5 data augmentation is explained as a technique which produces more images for minor classes than for majority classes. So data augmentation leads to

Table 4: Table with the number of images per class and dataset type if all images are used

Class	Training	validation	Test
0	16130	3226	6452
1	1527	305	611
2	3307	662	1323
3	546	109	218
4	443	88	177

Figure 8: Displaying the distribution of the data across training, validation and test data within the 5 classes when all images are used.



the case that in all classes the same number of images is reached. This can also be argued as a form of oversampling.

The number of test data will not be changed contrary to the number of training and validation data. This is to make results more comparable because e.g. the distribution of the data points per class in the test set has a huge impact on the results.

Pre-processing and Augmentation

In the pre-processing step all images are first resized to the dimension of (512×512), because all images of the dataset differ in the sizes. Because different pre-trained neural network architectures have different input sizes, the images are resized again before training the network depending on the selected architecture. For the Inception V3 architecture the image input size is 299.

Afterwards hist equalization is applied (see Chapter 4). Because the images are equalized in their depending colour histogram, but still differ in colour schemes, the images are then normalized. This means that the mean and the standard deviation is calculated for every color channel over the images in the training set. Because of computational resources only a maximum of 1000 images per class is used. The mean $\bar{x}$ and the standard deviation $sd(x)$ are then applied to every image in all datasets, including validation and test data. The following formula is used:

$$x_i = \frac{x_i - \bar{x}_i}{sd(x)_i}$$

(torch, 2019), where i defines the colour channel. These are all steps in the pre-processing of the images.

Augmentation of the images can be applied online and offline (see Chapter 4). As the goal of this thesis is to reduce the size of the training dataset, it is preferable to have control over the number of images used for training including the augmented images. Another already discussed problem is the imbalance of the dataset. Offline data augmentation can be used as an oversampling technique to balance the dataset. One possible way is to define an overall number of images for all classes and uniformly distribute this value to the classes. The difference between this number and the number of original images in one class is the number of augmented images per class.

On the other side computation time is increasing exponentially using offline augmentation. Because cross validation and bootstrapping, which is used in smaller training sets, is already leading to extended training time, online augmentation is a valid alternative. This technique leads to much more training iterations for different training set sizes and more adjustment possibilities in the training process. As the test accuracy will not decrease greatly compared to former related work (see Chapter 6), online augmentation is used. The techniques are random rotations, vertical flips, shearing the images and color jittering (see Chapter 4).

Settings

An extract of the settings file to train the neural network can be found in Table 11 in the appendix. Training set sizes will decrease after the rules seen in Table 5. The maximum number of training images is 29270.

Table 5: Reduction rules for training data starting with maximum possible images and ending with 50 images

Nr. Training Data	Decrease by
$20000 < n$	3000
$10000 < n \leq 20000$	2000
$1000 < n \leq 10000$	1000
$100 < n \leq 1000$	100
$n \leq 100$	20

For a very low number of images in the training dataset, the goodness of fit depends a lot on which data is selected. This has to be differentiated between two cases. The first case states which data points are generally selected as training data. Bootstrapping is a technique that uses random samples from the base dataset (Efron et al. 1993). This means that in n iterations images are randomly selected as training data. Afterwards the mean and standard deviation of the accuracy measures are calculated. This gives a more robust estimation of the fit of the trained neural network to the data. A larger variance in the bootstrapping results can also indicate that there are data quality problems of the diabetic retinopathy set (Lam et al. 2018). The bootstrapping technique is only used for smaller datasets, as for large datasets this technique consumes a lot of computational resources particularly time. The threshold of 1000 images defines the number under which bootstrapping is used. In this case 10 bootstrap iterations are used.

The second case is that in each bootstrap iteration specific images are used as training images and the training set is again split into training and validation data. The goodness of fit depends now on which images are used as training or validation data and can vary a lot. One possible technique to address this problem is cross validation. Cross validation means that the training data is repeatedly split into training and validation set. The split can be done using different techniques. Here in most cases the training data is split into six equal sized datasets. One part is used as validation set, the other five are used as training data. The number of splits is set to six as this is the general ratio of training and validation data. The splits are usually selected independent of the classes. Stratified k-fold cross validation considers the distribution of the response variable and adjusts the random sampling of the images to the overall distribution of classes in the response variable. This ensures that the imbalance of the dataset is transferred to the different cross validation splits. For Stratified k-fold cross validation with $k = 6$ the data size in each of the six splits needs to be equal or larger than six images. E.g. if only ten images are selected for training, then this constraint is not fulfilled. For these cases the splits are simply shuffled with the initial distribution in training and validation set. The number of random splits is also six. The best performing model in terms of Cohen's kappa is used as the output model of cross validation iterations (Fahrmeir et al. 2013).

Generally the number of cross validation iterations can be higher than the ones that are used here. But as usual in this thesis computational time is so high, that the number of bootstrap and cross validation iterations has to be limited.

Generally the input size of images into the inception network is 299×299 pixels. This number is increased to 512$\times$512, if the number of original data in the training dataset is smaller or equal than 500. This should increase the ability of the neural network to extract valuable information. This is only done for smaller training set sizes because it increases the number of parameters of the neural network from 24357354 to 29863914. The numbers are calculated using the python library torchsummary (Chandel, 2018).

Fine Tuning

As described in Chapter 4 transfer learning or fine tuning differentiates by the number of layers or weights that are retrained. For small datasets transfer learning is the better choice as only the last layer is trained. For larger datasets fine tuning should lead to better results. In this thesis, fine tuning is used with the exception that the first 5 Convolutional layers are not retrained and retraining starts with the first Inception layer.

The chosen architecture for training is version 3 of the Inception network. One characteristic of this network is the auxiliary output after Inception layer number 8. This layer has to be recognized in transfer learning because not only the final classifier, but also this auxiliary classifier has to be replaced by a fully connected layer with the number of classes of the new dataset. The outputs of both classifiers are summed up to calculate the loss in the training process. The final classification of an image depends only on the final classifier and not auxiliary layer.

Optimization and Adaptive Learning Rate

Classically Stochastic Gradient Descent (see Chapter 4) is used to optimize the weights in neural networks, but in recent years Adam (see Chapter 4) is often used now. In Ruder (2016) it is suggested to use Adam optimizer in problems where large data and/or complex and deep networks are used. In this thesis the Adam optimizer is used with the default values except for the weight decay which is set to 0.001.

Also a scheduler for the learning rate is implemented. In Hsueh et al. (2018) a new scheduler for learning rates is introduced and compared with e.g. step decay. The Adam optimizer is already adapting the learning rates, but the neural networks converge faster and to better goodness of fit with a scheduler applied to the learning rate than without a scheduler. This scheduler restates the learning rate every selected epoch to a factor of the initiate learning rate. In these experiments the factor is 0.1 and the step size is 12.

In Chapter 3 the problem is discussed if the data hints to a classification or an ordinal regression model. It is clear that the classes that state the DR are ordered, but it is not clear that the distance between the different classes is known and especially not clear to be one as the class labels would state. Also in various experiments the results of classification modelling are better than regression modelling for the diabetic retinopathy dataset. In this thesis the problem is seen as an classification problem and Cross Entropy loss function is used. Another problem of the dataset is the imbalance in the classes. As in Chapter 5 already explained the loss function will be weighted to adapt to the imbalance in the classes. The oversampling technique is discussed in Chapter 4 and 5 as a possible solution to this problem.

Batch Size

The batch size depends on the overall amount of images. For a large number of training data larger batch sizes are recommendable, because in one batch the neural network will train on more data and so has less variance in single epochs. Downside of larger batches is that training takes longer and consumes more memory. The used batch sizes based on the number of training images can be seen in Table 6.

Table 6: Mini-batch sizes depending on the number of overall images n in training process

Nr. Training Data	Batch size
$0 < n \leq 50$	5
$50 < n \leq 1000$	32
$500 < n \leq 2000$	256
$2000 < n \leq 10000$	1024
$10000 < n$	2048

Early Stopping

Early Stopping is used to prevent overfitting. When Cohen's kappa score is improving in the training process, but decreasing on the validation data, then the training process will be stopped. In literature no general number of epochs exist, after which the training process should be stopped. In this thesis the training process is stopped when eight successive epochs could not provide a better Cohen's kappa score on the validation set. This should be appropriate because for a smaller number of data it does not lengthen the training time extensively and also does not stops the process too early when the training got stuck in a local minima.

6 Results

In this section the results are presented. First a look is taken at the results when all images are used to train the convolutional neural network. Afterwards the number of images in the training set is successively reduced. Out of the first results, adjustments on the settings and distribution of data is taken and the results are evaluated again.

All Data

The aim is to reduce the training set sizes, while still keeping the Cohen's kappa score as high as possible. In other words: The goal is to find a benchmark value for Cohen's kappa score and then see how stable the predictions are for less data and if there is a value after which a drop-off of Cohen's kappa score can be observed. This means that it is important to first optimize the results for the case when all training data is used and use this as a benchmark score. In Table 7 the results can be seen when all images are used for training a neural network. Butterworth et al. 2016 is the paper with most comparable settings and the stated test accuracy and kappa score are 76.6% and 0.651 there. Compared with the results from that paper the test accuracy is slightly lower. Whereas the kappa score is a lot lower. Differences can occur due to multiple reasons. First in Butterworth et al. 2016 the messidor dataset is additionally used. Second ResNet instead of the Inception architecture is used. Also a different split into test and training images is probable because of the additional images. The hyperparameter settings will differ as well, e.g. in this thesis the first convolutional layers are not retrained due to computational resources reasons.

To get a better and a more differentiated impression of the results the confusion matrix and the sensitivity per class can be found in Figure 9 and Table 8. Sensitivity can be interpreted as the correct predictions per class per true label. The most obvious observation is that for class 0, so if there is no DR, the prediction accuracy is really good. For the classes 2 and 4 the number of correct predicted images is higher than in the remaining category but is not higher than 50%. The worst prediction

Table 7: Goodness of fit measures on test set if all images are used for training

Measure	Value
Accuracy	0.7285
Top 2 Accuracy	0.9063
Cohens Kappa Score	0.3372
MCC	0.3404
MAE	0.4594

rate appears in class 1. Here 472 of 606 images are classified into class 0. This means 77% of the images are seen as there is no DR, which could mean that it is really hard to differentiate between class 0 and class 1. A similar but not so extreme picture can be observed in class 3 where almost 55% of the images are classified into class 2.

Both cases are probably leading towards the higher top-2 accuracy of 90.63% because it is likely that the second most probable prediction is the actual true label.

Tables 12 and 13 in the appendix strengthen this picture. The tables show that most images are predicted into class 0 and 2. The numbers of images in class 0 and 2 are higher for predicted images than for the true classes. For all classes the main share of images is generally predicted to be class 0.

Figure 9: Confusion Matrix of true and predicted labels in the network with the best Cohen's kappa score. The network is trained using the maximum available number of images.

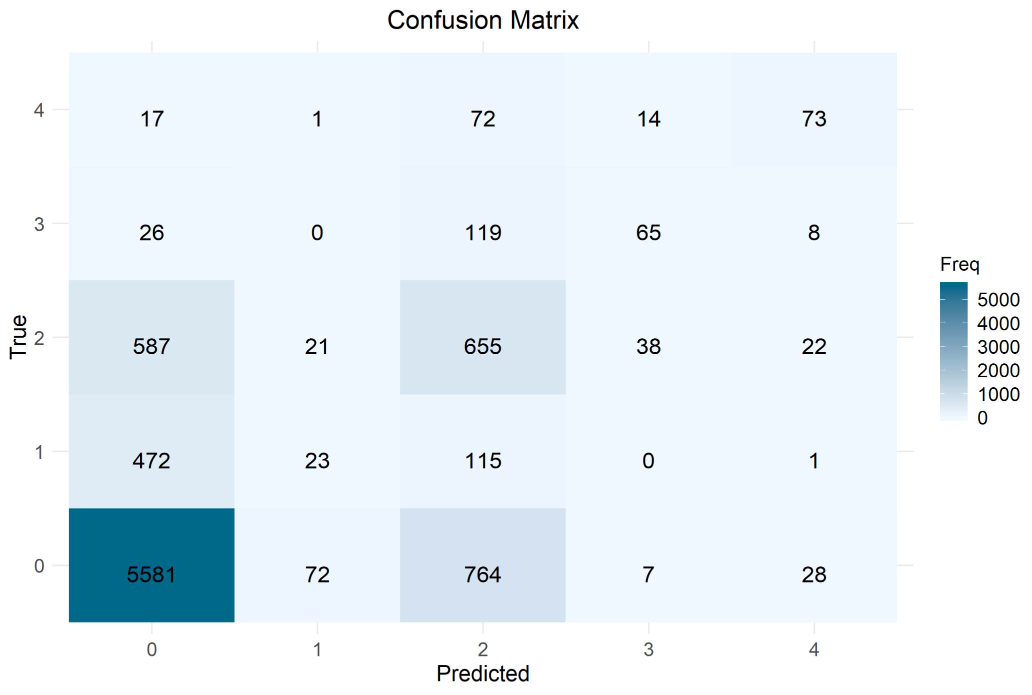


Table 8: Sensitivity per class if all images are used for training

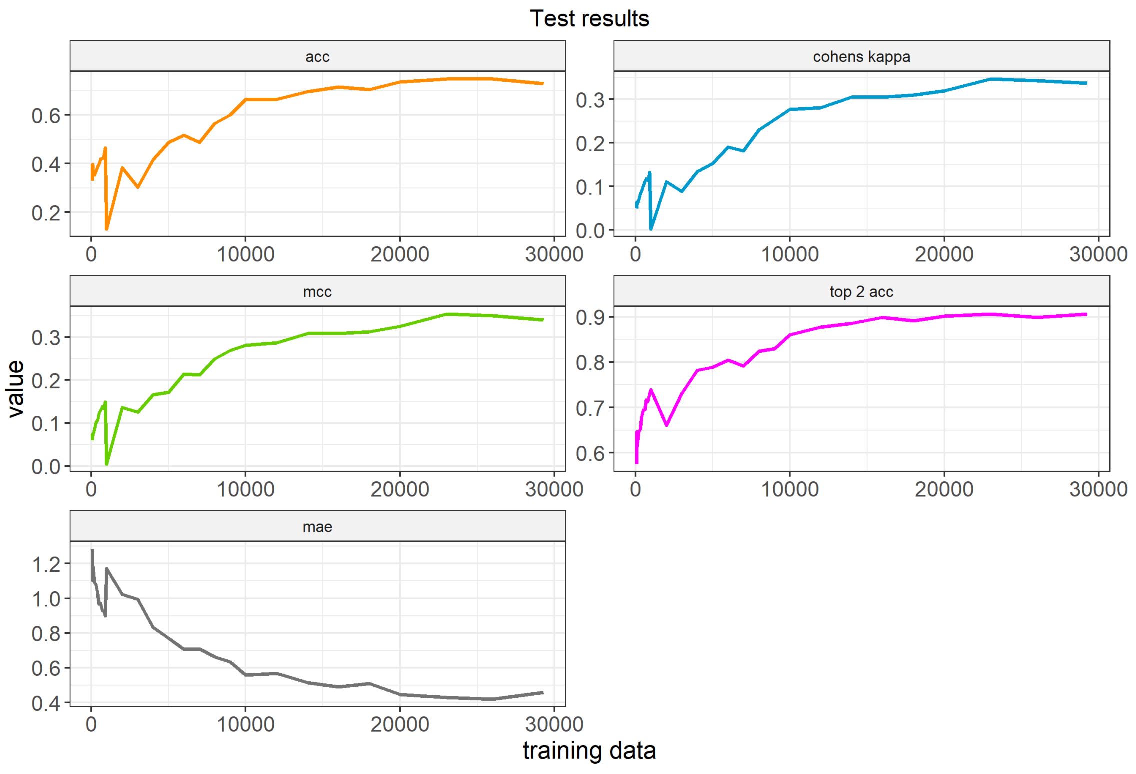
Class	Sensitivity
0	0.86
1	0.04
2	0.50
3	0.30
4	0.41

Reduce Training Data

In Chapter 5 the number of images that are used as training data is defined and in Table 5 in Chapter 5 the rules for the reduction of the data is explained. A combination of bootstrapping and cross validation is used when less or equal than 1000 images are used for training the neural network. As training a neural network is computationally really heavy and time consuming the bootstrap iterations have been limited to 10 and cross validation is done using k-fold cross validation with 6 splits as only 1/6 is used as validation data. The results can be seen in Figure 10. The measures indicate a stable goodness of fit until the mark of 10000 training images is reached. Afterwards all measures decrease strongly.

The most important measures are accuracy and especially Cohen's kappa score. Interestingly Cohen's kappa score is not really high with a max value around 0.3. The values decrease to a minimum value of 0. This indicates that the level of agreement between true and predicted values is not really high. A look at the accuracy show similar results. In a maximum of 73% of the cases the neural network predicts the correct state of DR. In combination with the imbalance of the data and that over 73% of the images are in class 0, this is not really high. The top-2 accuracy indicates that in many cases the second most probable prediction is correct as here a value of

Figure 10: Progress of the coefficients used for measuring the goodness of fit with decreasing number of images used for training the neural network.

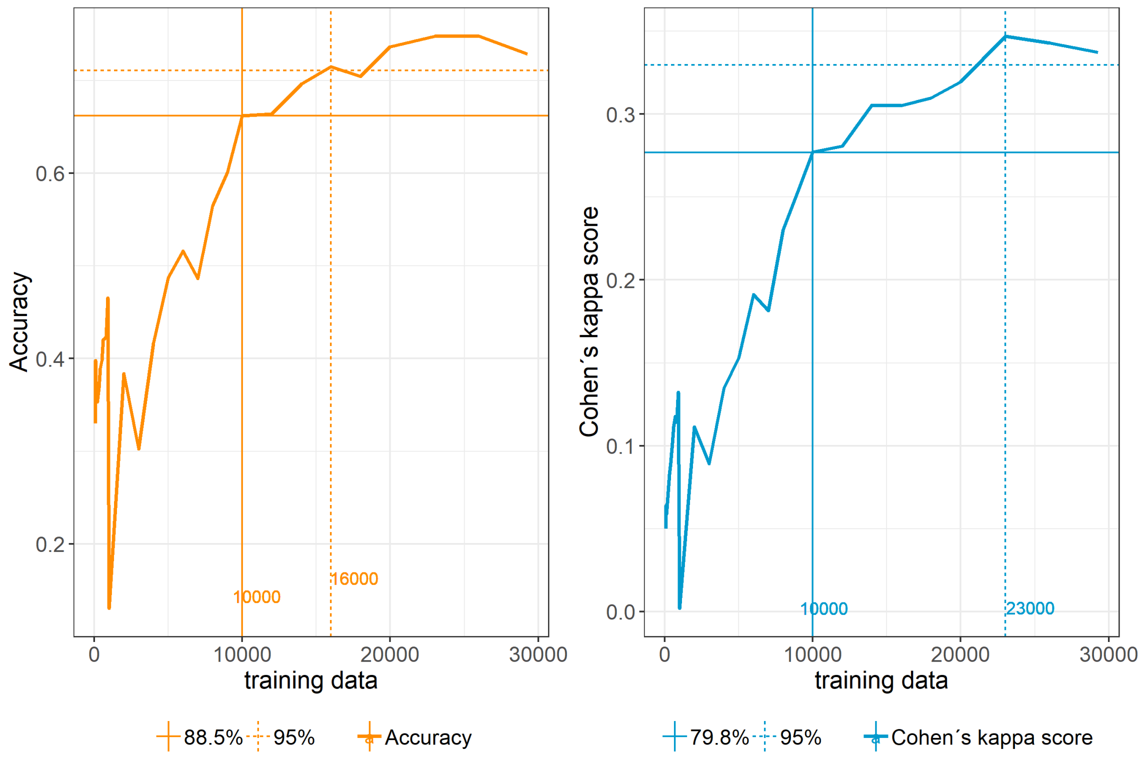


over 90% can be reached. Even in the worst case here a accuracy of nearly 60% is reached. Matthews correlation coefficient has nearly the same curve and values as Cohen's kappa score. The minimum values of the mean absolute error are slightly below 0.5. The maximum value is at 1.2. This means that on average the difference between the true and predicted value is more than one class.

In Figure 11 only the accuracy and Cohen's kappa score are shown with decreasing number of images. The vertical dotted lines display the minimum number of images that has to be used that 95% of the maximum accuracy or Cohen's kappa score is reached. These values differ and are 16000 images for the accuracy and 23000 for Cohen's kappa score. The other vertical line is at 10000 images and in the legend, it can be seen that for the accuracy this means that with 10000 images 88% of the maximum accuracy are reached. For Cohen's kappa score only 79% of the maximum are reached. For Cohen's kappa score, this is influenced a lot by the fact that for 23000 images a value much higher than any other value is reached and afterwards a larger drop-off can be seen.

In both figures (10 and 11) an unusual development is that the coefficients increase in

Figure 11: Progress of accuracy and Cohen's kappa score with decreasing number of images used for training the neural network. Vertical Lines show thresholds which show the number of images that has to be reached so that the a given percentage (see legend) of the maximum coefficient value is reached.



terms of the goodness of fit at the 1000 images mark. This can be explained with the usage of cross validation and bootstrapping. Cross validation takes the best result out of the six splits and is averaged over the bootstrap iterations.

The sensitivity per class is calculated and displayed in Figure 12. Above 10000 images the sensitivities per class are stable. Under 10000 images the sensitivities per class are changing heavily. Between 1000 and 10000 images the sensitivities per class are heading towards the value of 0.5, so towards each other. They are either decreasing like for class 0 or increasing like for class 1. So high sensitivities are decreasing and low ones are increasing. This can be explained due to the distribution of the training data by class. In Table 9 the distribution of images across datasets and classes is shown for a combined 6000 images in the training and validation set. Here the number of images is equal for classes 0, 1, and 2, so the distribution of data per class is more uniform in comparison to the imbalance of the total dataset (see Table 4 on page 38).

In Figures 10, 11 and 12 it is very hard to properly observe what is happening when less than 1000 images are used. In the appendix in Figure 18 and 19 the values for Cohen's kappa score, accuracy and sensitivities per class are displayed with the filter that less than 1000 images are used as training data. The effect of cross validation

Figure 12: Progression of the Sensitivity per class with decreasing number of training images.

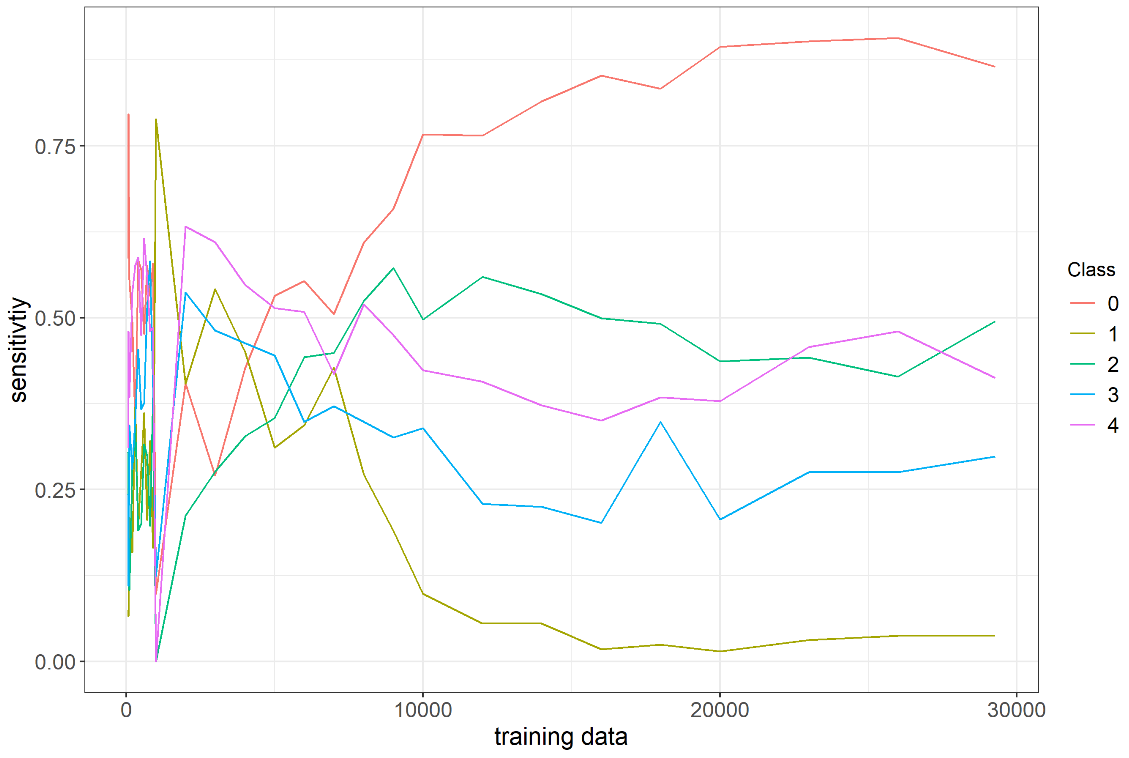


Table 9: Counting number of images per class if 6000 images are used as training data

Class	Training	Validation	Test
0	1338	267	6452
1	1338	267	611
2	1338	267	1323
3	546	109	218
4	443	88	177

can be seen clearer when observing the sensitivities per class in Figure 19. The values here are much more similar to the values for large training sets than for mediocre sets. The sensitivities for class 0 and 4 give the highest values. One important notice here is, that not only the best model of the cross validation, but also of all bootstrap iterations is used for the sensitivities. More reliable results for less than 1000 images are the accuracy and the Cohen's kappa score that are aggregated over the bootstrap iterations. These show that cross validation improves the test results and that the performance on unseen data still declines.

Correction of training data distribution and cross validation

Two main conclusions can be taken out of the above results. These are that the imbalance has a huge influence on the prediction accuracy and that cross validation is increasing the performance on test data. Out of these conclusions two changes are taken. First is that the imbalance should be transferred two smaller training set sizes. The procedure for this is that the percentage of images per class is calculated if all images are used and is applied to smaller training set sizes. The percentage per class can be seen in Table 1 on page 10. For really small training set sizes this would lead to zero images in classes 3 and 4. So one exception is that at least one image has to be used per class in training and validation set. This exception can lead to minor changes in the distribution, but should not influence the results too much.

The other change is that for training set sizes with less or equal than 10000 images instead of 1000 images cross validation and bootstrapping is applied. Cross validation is done in the same way as before, which means that the training set is split into 6 subsets and successively one part is used as validation set. The numbers of bootstrap iterations are shown in the appendix in Table 14. This means that for images between 1000 and 10000 images only cross validation is used and bootrapping will only be used for less than 1000 images.

The results are shown in Figure 13. The peak performance is the same as before. This is expected because the distribution of the data does not change much when a high number of images is used. Surprisingly the performance even increases when using 18000 to 23000 images. Compared to the results in Figure 10 the values of the coefficients are more stable. This should indicate that the performance on less training data is better when the distribution of the training data per class is considered. An exceptional performance value occurs when 2000 images are used as training data. Interestingly the accuracy, top-2 accuracy and the mean absolute error show really good performance, whereas the mcc shows little improvement and for the Cohen's kappa score the value is as expected. This shows very well that the MCC and Cohen's kappa score are more reliable coefficients than the other measures for the imbalanced dataset.

In Figure 14 the accuracy and Cohen's kappa score are displayed. The vertical lines show again a number of images where the given percentage of the maximum value of the measure is reached. Here the improvements are obvious. For the accuracy a value of 95% of the maximum reached accuracy score is reached when 7000 images are used. In Figure 11 this value is 16000 images. 10000 images were resulting in 88.5% whereas now even with 3000 images 90% are reached if the value for 2000

Figure 13: Progress of the coefficients used for measuring the goodness of fit with decreasing number of images used for training the neural network.

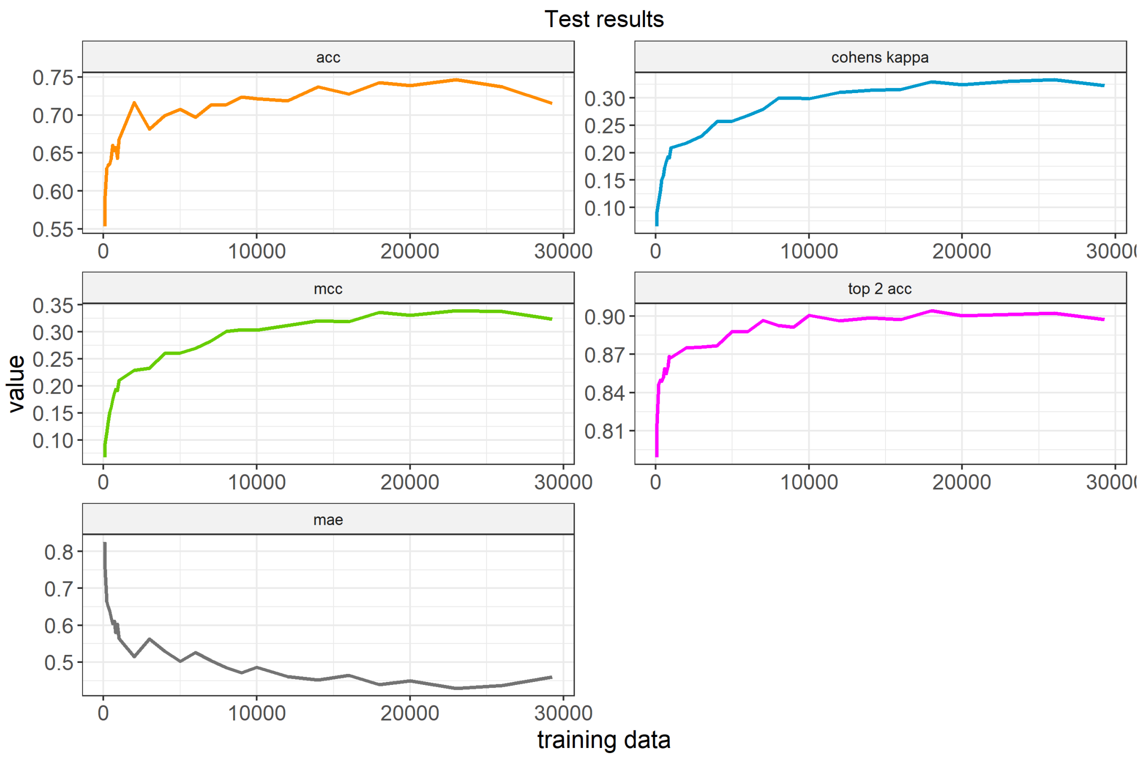
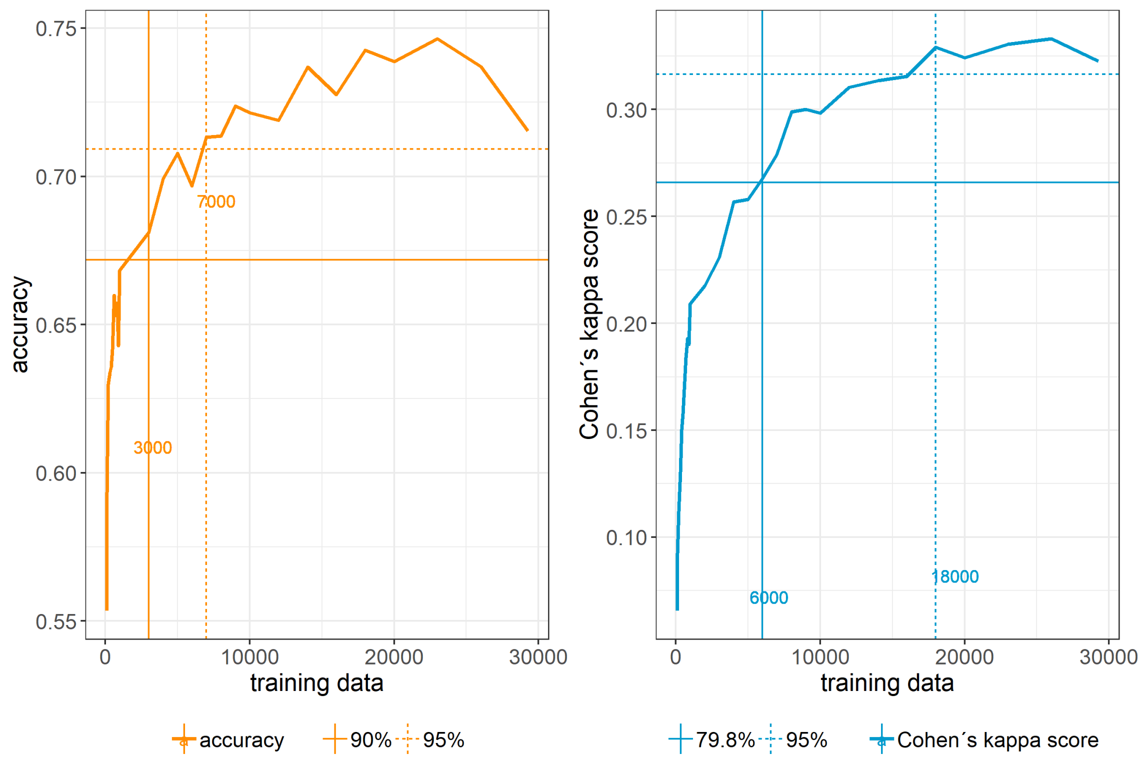


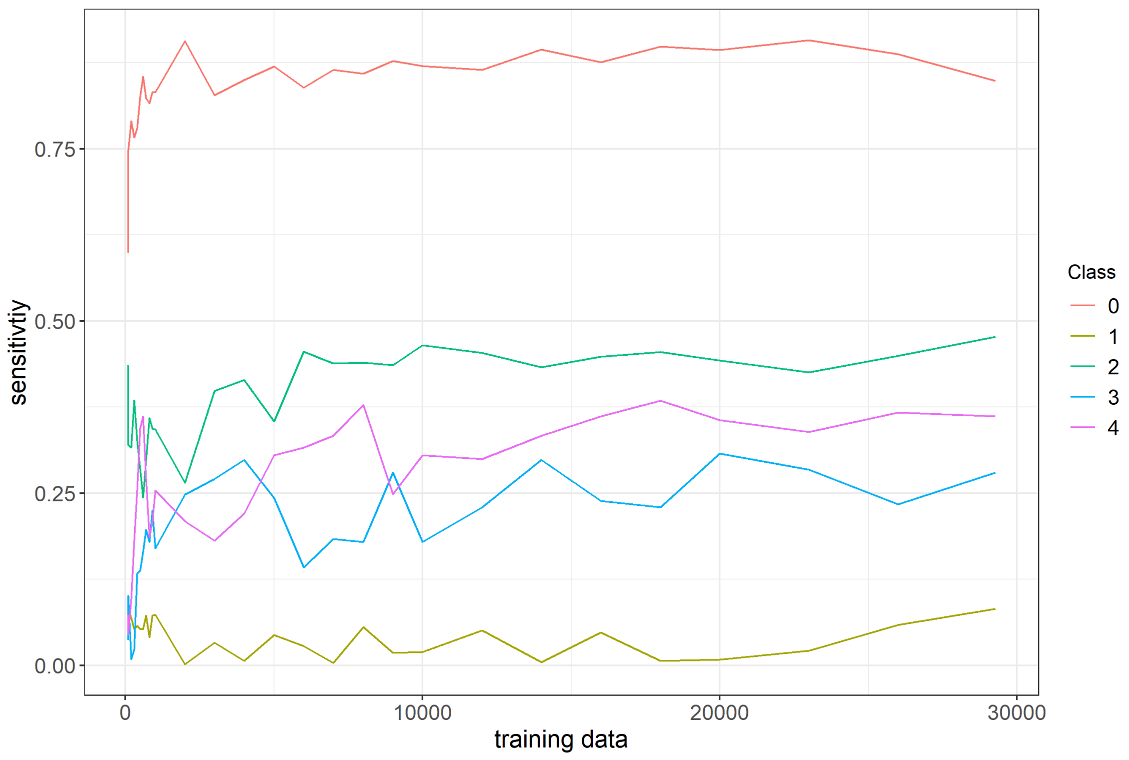
Figure 14: Progress of accuracy and Cohen's kappa score with decreasing number of images used for training the neural network. Vertical Lines show thresholds which show the number of images that has to be reached so that the a given percentage (see legend) of the maximum coefficient value is reached.



images is seen as an outlier. For Cohen's kappa score a similar picture shows up even if the results are not as good as for the accuracy. 95% of the maximum Cohen's kappa score is reached using 18000 images compared to 23000 images in Figure 11. 10000 images delivered a percentage of 79.8 of the maximum value before. This value is reached with 6000 images now.

Is there an explanation for this improvement? A look at the progress of the sensitivities per class could help here. In Figure 15 these sensitivities are shown and indicate the exact purpose of why the changes in the distribution of the training dataset is made. The lines are much more stable even in smaller training data regions compared to the sensitivities in Figure 12. Particularly the sensitivity of class 0 stays really high. Because about 73% of the images are in class 0, a high sensitivity for class 0 results in a high overall accuracy. An outlier is recognised for 2000 training data images. This outlier can be explained through this figure because the sensitivity for class 0 is really high. The Cohen's kappa score is not affected by this because the sensitivity for class 2 and class 3 are dropping off here. So in this case the neural network is able to learn patterns really good for class 0 but not for the other classes.

Figure 15: Progression of the Sensitivity per class with decreasing number of training images.



Threshold

The progression of the measures is discussed in detail above. The last open question is if it is possible to define a threshold which defines a minimum number of images that should be used for training a neural network. One approach is to find or define a value like 95% or 90% of the maximum score. The threshold should not come below this value. In Figure 16 the ratios between the Cohen's kappa scores and the maximum kappa scores are drawn. For 9000 images 90% of the maximum score is reached. Arguably 8000 images should be appropriate and reasonable, too, since the Cohen's kappa is only slightly below the 90% mark of the maximum Cohen's kappa score. For 18000 images 95% is reached.

Computation Times

One big limitation in training neural networks is that huge compuational resources are needed. In this thesis, especially time is a big limitation because cross validation and bootstrapping are used. In the left plot of Figure 17 the training times per training set size are shown. They obviously decrease with decreasing images. The ratio between training and testing time is very high for large training set sizes. For a decreasing number of training images the ratio between training time and testing

Figure 16: The ratios between Cohen's kappa score and the maximum score.

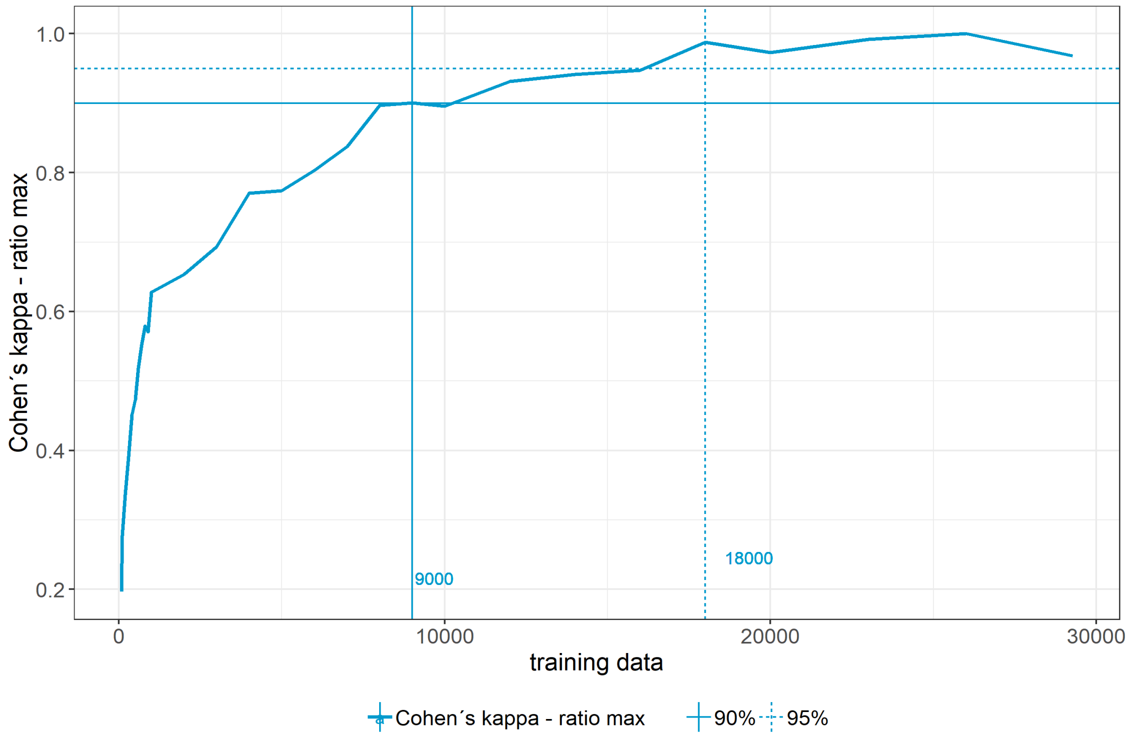
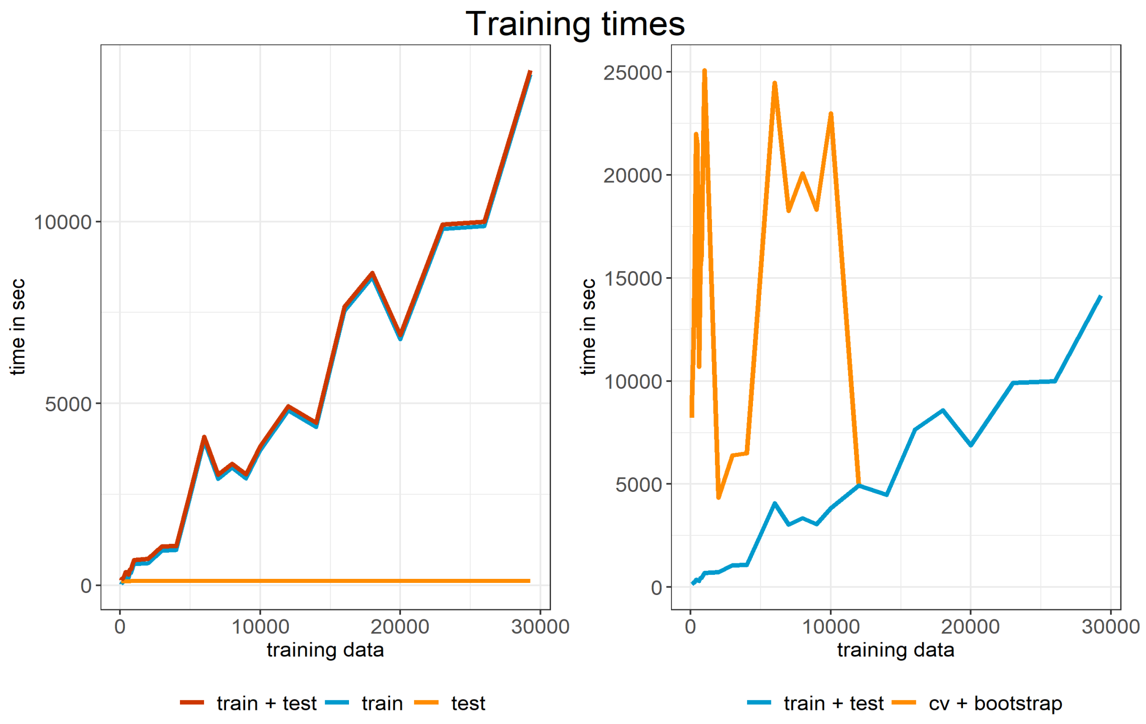


Figure 17: Times to train the neural networks per training set size. Left: Differentiation between training, test and combined time for only one run. Right: Estimation of total time taking cross validation and bootstrapping into account.



time is decreasing. Important is that cross validation and bootstrapping are not considered here and only one training run is taken into account. The right plot shows the times which are adjusted by number of cross validation and bootstrap iterations. These are only estimated based on one run.

In Table 10 the times are converted to days, hours, minutes and seconds to really understand how long training neural networks is taking. To train neural networks with the different number of images in the training set in combination with cross validation and bootrapping takes over 4 days.

Table 10: Times for training the neural networks per train data size. Times for cross validation and bootstrapping are estimated by one run.

training data	Time	Time adjusted
29270	3h 56min 5sec	3h 56min 5sec
26000	2h 46min 37sec	2h 46min 37sec
23000	2h 45min 17sec	2h 45min 17sec
20000	1h 54min 45sec	1h 54min 45sec
18000	2h 23min 10sec	2h 23min 10sec
16000	2h 7min 36sec	2h 7min 36sec
14000	1h 14min 33sec	1h 14min 33sec
12000	1h 22min 7sec	1h 22min 7sec
10000	1h 3min 51sec	6h 23min 4sec
9000	0h 50min 54sec	5h 5min 27sec
8000	0h 55min 45sec	5h 34min 30sec
7000	0h 50min 43sec	5h 4min 18sec
6000	1h 7min 60sec	6h 47min 57sec
5000	0h 42min 48sec	4h 16min 45sec
4000	0h 18min 3sec	1h 48min 16sec
3000	0h 17min 47sec	1h 46min 43sec
2000	0h 12min 4sec	1h 12min 26sec
1000	0h 11min 37sec	6h 58min 8sec
900	0h 9min 60sec	5h 59min 51sec
800	0h 7min 28sec	4h 29min 3sec
700	0h 7min 10sec	4h 18min 12sec
600	0h 4min 57sec	2h 58min 17sec
500	0h 5min 59sec	5h 59min 3sec
400	0h 6min 7sec	6h 6min 41sec
300	0h 4min 30sec	4h 29min 38sec
200	0h 3min 18sec	3h 18min 5sec
100	0h 2min 29sec	2h 29min 15sec
80	0h 2min 17sec	2h 16min 48sec
total	1d 1h 55min 57sec	4d 9h 52min 38sec

7 Summary

In this thesis colour fundus images, which are images of the human eye, are classified into stages of diabetic retinopathy (DR). The dataset used for this is a former Kaggle challenge dataset with 35124 images labelled into five classes which state the DR of an eye. One main characteristic of the dataset is the imbalance, because class 0 holds about 73% of the data points. The aim of the thesis is to use convolutional neural networks to predict the state of DR of an eye image and reduce the training dataset successively to find a threshold below which the training dataset is too small to train a neural network and achieve good performance.

This is a typical use case of retraining already pre-trained neural networks. An inception v3 convolutional neural network is used. The network is pre-trained with the imagenet dataset (Deng et al. 2009). The training dataset is successively reduced and used to retrain the neural networks. Batch sizes are adapted depending on the training dataset. A maximum accuracy of 73% is reached which is comparable to results of papers with similar use cases such as Butterworth et al. (2016). Cohen's kappa score which is used as the performance measure in the Kaggle challenge indicates with a value of 0.35% a mediocre fit of predicted and true labels. When reducing the training set the results are stable with larger training set sizes until the training set size of 10000 images is reached. Afterwards the prediction accuracy measures are decreasing. As a threshold 9000 images should at least be used to train a neural network. The condition for this threshold is that a minimum of 90% of the maximum Cohen's kappa score on the test data should be reached.

Discussion

Two noticeable observations are that results increase in terms of accuracy and Cohen's kappa score when cross validation is used and that the sensitivity per class is changing when the training dataset is getting smaller than 10000 images. The settings are adjusted so that the imbalance across classes is transferred to all training sizes and cross validation is applied when using less than 10000 instead of 1000 images. The changes are resulting in more stable prediction accuracies. Especially the sensitivities per class (Figure 15) show that the distribution of the training data across classes is important for good results. The reason for this is mainly because the neural network can train the patterns for class 0 better as there are more images of this class. This leads to a better sensitivity of class 0 and so to a better overall accuracy.

Also cross validation leads to better results in smaller training datasets. The predictions are much better for training set sizes of 1000 to 10000 images. Based on these findings a threshold for the number of images that should be at least be used is debatable. But 9000 images should be a reasonable number that would lead to appropriate results.

Outlook

A problem that is occuring when analysing this dataset and classify images is at first the imbalance of the dataset. Classical techniques to overcome this issue are weighting of the loss function, which is used in this thesis, oversampling and undersampling. Oversampling is the most promising technique (see Buda et al. (2018)) and can be used via offline data augmentation techniques. But this technique would extend computation times heavily, because a lot more training images would be used. Therefore this technique is not used. In future work this is a technique that could lead to better results particularly for smaller datasets.

Another improvement can be bootstrapping that is applied only rarely because of computation time, too. Bootstrapping means that for smaller training data sizes bootstrapping can make the results more stable and could lead to better peak results, because there are more iterations where different images are used as training data.

The major question of this thesis is if the neural network is able to classify images when the dataset is reduced or small. Neural networks are at their best when a lot of data points are used. Other techniques such as Bayesian classifier could lead to a better performance when smaller datasets are used. In Sudha et al. (2018) promising results are achieved on a binary classification problem for diabetic retinopathy with only 385 data points. In Somfai et al. (2014) a Bayesian artificial neural network is used for the classification of a small diabetic retinopathy dataset. Also Deep Learning can be combined with Bayesian uncertainty measures as in Leibig et al. (2017). Additionally support vector machines (Faisal et al. 2014) could be a promising technique for the classification of a few medical images.

References

Abràmoff, Michael D, Mona K Garvin, and Milan Sonka (2010). “Retinal imaging and image analysis”. In: *IEEE reviews in biomedical engineering* 3, pp. 169–208.

Aggarwal, Charu C. (2018). *Neural Networks and Deep Learning. A Textbook.* 1st ed. Springer International Publishing. ISBN: 978-3-319-94463-0.

Alban, Marco and Tanner Gilligan (2016). “Automated detection of diabetic retinopathy using fluorescein angiography photographs”. In: *Report of standford education.*

Antony, Mathis and Stepan Brüggemann (2015). *Kaggle Diabetic Retinopathy Detection; Team o_O solution.* Competition Report Github. URL: `https://github.com/sveitser/kaggle_diabetic/blob/master/doc/report.pdf`.

Beckham, Christopher and Christopher Pal (2017). “Unimodal probability distributions for deep ordinal classification”. In: *arXiv preprint arXiv:1705.05278.*

Berrada, Leonard, Andrew Zisserman, and M Pawan Kumar (2018). “Smooth Loss Functions for Deep Top-k Classification”. In: *arXiv preprint arXiv:1802.07595.*

Bloice, Marcus D, Christof Stocker, and Andreas Holzinger (2017). “Augmentor: an image augmentation library for machine learning”. In: *arXiv preprint arXiv: 1708.04680.*

Breiman, Leo et al. (2001). “Statistical modeling: The two cultures (with comments and a rejoinder by the author)”. In: *Statistical science* 16.3, pp. 199–231.

Buda, Mateusz, Atsuto Maki, and Maciej A Mazurowski (2018). “A systematic study of the class imbalance problem in convolutional neural networks”. In: *Neural Networks* 106, pp. 249–259.

Butterworth, David T, Shohin Mukherjee, and Mohit Sharma (2016). “Ensemble Learning for Detection of Diabetic Retinopathy”. In: *30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.*

Chandel, Shubham (2018). *torchsummary 1.5.1: Model summary in PyTorch similar to ‘model.summary()‘ in Keras.* [Online accessed 2019-01-05]. URL: `https://github.com/sksq96/pytorch-summary`.

Cheng, Jianlin, Zheng Wang, and Gianluca Pollastri (2008). “A neural network approach to ordinal regression”. In: *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on.* IEEE, pp. 1279–1284.

Chicco, Davide (2017). “Ten quick tips for machine learning in computational biology”. In: *BioData mining* 10.1, p. 35.

Cohen, Jacob (1960). “A coefficient of agreement for nominal scales”. In: *Educational and psychological measurement* 20.1, pp. 37–46.

Cohen, Jacob (1968). "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." In: *Psychological bulletin* 70.4, p. 213.

Dahl, David B. et al. (2018). *xtable: Export Tables to LaTeX or HTML*. R package version 1.8-3. URL: `https://CRAN.R-project.org/package=xtable`.

Decencière, Etienne et al. (2014). "Feedback on a publicly distributed database: the Messidor database". In: *Image Analysis & Stereology* 33.3, pp. 231–234. ISSN: 1854-5165. DOI: `10.5566/ias.1155`. URL: `http://www.ias-iss.org/ojs/IAS/article/view/1155`.

Deng, J. et al. (2009). "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

Efron, Bradley and Robert J Tibshirani (1993). *An introduction to the bootstrap.* Chapman & Hall, New York.

EYEPACS, LLC (2018). *EyePacs: Picture Archive Communication System.* [Online accessed 2018-10-05]. URL: `http://www.eyepacs.com`.

Fahrmeir, Ludwig, Thomas Kneib, Stefan Lang, and Brian Marx (2013). *Regression: Models, Methods and Applications.* Berlin: Springer-Verlag.

Faisal, Muhammad et al. (Jan. 2014). "Classification of diabetic retinopathy patients using support vector machines (SVM) based on digital retinal image". In: *Journal of Theoretical and Applied Information Technology* 59, pp. 197–204.

Fawcett, Tom (2006). "An introduction to ROC analysis". In: *Pattern recognition letters* 27.8, pp. 861–874.

Gardner, GG, D Keating, Tom H Williamson, and Alex T Elliott (1996). "Automatic detection of diabetic retinopathy using an artificial neural network: a screening tool." In: *British journal of Ophthalmology* 80.11, pp. 940–944.

Gargeya, Rishab and Theodore Leng (2017). "Automated identification of diabetic retinopathy using deep learning". In: *Amercian Academy of Ophthalmology* 124.7, pp. 962–969.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). *Deep learning.* Vol. 1. MIT press Cambridge.

Graham, Ben (2015). "Kaggle diabetic retinopathy detection competition report". In: *University of Warwick.*

Gulshan, Varun et al. (2016). "Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs". In: *JAMA - Journal of the American Medical Association* 316.22, pp. 2402–2410.

Hartnett, M Elizabeth, Wolfgang Baehr, and Yun Zheng Le (2017). "Diabetic retinopathy, an overview". In: *Vision Research* 139, pp. 1–6.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hsueh, Bo Yang, Wei Li, and I-Chen Wu (2018). “Stochastic Gradient Descent with Hyperbolic-Tangent Decay”. In: arXiv: `1806.01593v1 [cs.CV]`.

Ioffe, Sergey and Christian Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv: 1502.03167*.

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani (2013). *An introduction to statistical learning: with applications in R*. New York: Springer.

Jiang, Jianmin, Paul R. Trundle, and Jinchang Ren (2010). “Medical image analysis with artificial neural networks”. In: *Computerized medical imaging and graphics : the official journal of the Computerized Medical Imaging Society* 34 8, pp. 617–31.

Jones, Eric, Travis Oliphant, Pearu Peterson, et al. (2001). *SciPy: Open source scientific tools for Python*. [Online; accessed 2018-12-05]. URL: `http://www.scipy.org/`.

Kaggle (2018). *Kaggle*. [Online accessed 2018-10-05]. URL: `https://www.kaggle.com/`.

Kayalibay, Baris, Grady Jensen, and Patrick van der Smagt (2017). “CNN-based segmentation of medical imaging data”. In: *arXiv preprint arXiv:1701.03056*.

Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.

Lakhani, Paras et al. (2018). “Hello World Deep Learning in Medical Imaging”. In: *Journal of digital imaging* 31.3, pp. 283–289.

Lam, Carson, Darvin Yi, Margaret Guo, and Tony Lindsey (2018). “Automated Detection of Diabetic Retinopathy using Deep Learning”. In: *AMIA Summits on Translational Science Proceedings* 2017, p. 147.

Lee, June-Goo et al. (2017). “Deep learning in medical imaging: general overview”. In: *Korean journal of radiology* 18.4, pp. 570–584.

Leibig, Christian et al. (2017). “Leveraging uncertainty information from deep neural networks for disease detection”. In: *Scientific Reports*.

Litjens, Geert et al. (2017). “A survey on deep learning in medical image analysis”. In: *Medical image analysis* 42, pp. 60–88.

Liu, Yanzhu, Adams Wai Kin Kong, and Chi Keong Goh (2018). “A Constrained Deep Neural Network for Ordinal Regression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 831–839.

LRZ (2018). [Online accessed 2018-12-27]. URL: `https://www.lrz.de/`.

Makandar, Aziz, Anita Patrot, and Bhagirathi Halalli (2014). “Color Image Analysis and Contrast Stretching using Histogram Equalization”. In: *International Journal of Advanced Information Science and Technology (IJAIST)* 2319, p. 2682.

Matthews, Brian W (1975). “Comparison of the predicted and observed secondary structure of T4 phage lysozyme”. In: *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405.2, pp. 442–451.

McKinney, Wes (2010). “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman, pp. 51–56.

Micikevicius, Paulius et al. (2017). “Mixed precision training”. In: *arXiv preprint arXiv:1710.03740*.

More, Ajinkya (2016). “Survey of resampling techniques for improving classification performance in unbalanced datasets”. In: *arXiv preprint arXiv:1608.06048*.

Niu, Zhenxing et al. (2016). “Ordinal regression with multiple output cnn for age estimation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4920–4928.

Noronha, Kevin and K Prabhakar Nayak (2012). “A review of fundus image analysis for the automated detection of diabetic retinopathy”. In: *Journal of Medical Imaging and Health Informatics* 2.3, pp. 258–265.

NVIDIA (2017). *NVIDIA Tesla V100 GPU Architecture*. [Online accessed 2018-11-20]. URL: `http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`.

nvidia (2019). *Nvidia*. [Online accessed 2019-01-12]. URL: `https://www.nvidia.com/en-us/data-center/dgx-1/`.

O’Shea, Keiron and Ryan Nash (2015). “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458*.

Paszke, Adam et al. (2017). “Automatic differentiation in PyTorch”. In: *NIPS-W*.

Pratt, Harry et al. (2016). “Convolutional neural networks for diabetic retinopathy”. In: *Procedia Computer Science* 90, pp. 200–205.

Quellec, Gwenolé et al. (2017). “Deep image mining for diabetic retinopathy screening”. In: *Medical image analysis* 39, pp. 178–193.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: `https://www.R-project.org/`.

Razzak, Muhammad Imran, Saeeda Naz, and Ahmad Zaib (2018). “Deep Learning for Medical Image Processing: Overview, Challenges and the Future”. In: *Classification in BioApps*. Springer, pp. 323–350.

Reddi, Sashank J, Satyen Kale, and Sanjiv Kumar (2018). “On the convergence of adam and beyond”. In:

Rossum, Guido (1995). *Python Reference Manual.* Tech. rep. Amsterdam, The Netherlands.

Roth, Holger R et al. (2018). “An application of cascaded 3D fully convolutional networks for medical image segmentation”. In: *Computerized Medical Imaging and Graphics* 66, pp. 90–99.

Ruder, Sebastian (2016). “An overview of gradient descent optimization algorithms”. In: arXiv: `1609.04747v2 [cs.LG]`.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *nature* 323.6088, p. 533.

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

Santurkar, Shibani, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry (2018). “How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift)”. In: *arXiv preprint arXiv: 1805.11604*.

Scherer, Dominik, Andreas Müller, and Sven Behnke (2010). “Evaluation of pooling operations in convolutional architectures for object recognition”. In: *Artificial Neural Networks–ICANN 2010*. Springer, pp. 92–101.

Shen, Dinggang, Guorong Wu, and Heung-Il Suk (2017). “Deep learning in medical image analysis”. In: *Annual review of biomedical engineering* 19, pp. 221–248.

Simonyan, Karen and Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556*.

Somfai, Gábor Márk et al. (2014). “Automated classifiers for early detection and diagnosis of retinopathy in diabetic eyes”. In: *BMC Bioinformatics* 15.1, p. 106. ISSN: 1471-2105. DOI: `10.1186/1471-2105-15-106`. URL: `https://doi.org/10.1186/1471-2105-15-106`.

Sudha, V and C Karthikeyan (2018). “Analysis of diabetic retinopathy using naive bayes classifier technique”. In: *International Journal of Engineering & Technology* 7, p. 440. DOI: `10.14419/ijet.v7i2.21.12462`.

Suzuki, Kenji (2017). “Overview of deep learning in medical imaging”. In: *Radiological physics and technology* 10.3, pp. 257–273.

Szegedy, Christian, Wei Liu, et al. (June 2015). “Going deeper with convolutions”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9. ISSN: 1063-6919.

Szegedy, Christian, Vincent Vanhoucke, et al. (2016). “Rethinking the inception architecture for computer vision. 2015”. In: *arXiv preprint arXiv:1512.00567*.

Tajbakhsh, Nima et al. (2016). “Convolutional neural networks for medical image analysis: Full training or fine tuning?” In: *IEEE transactions on medical imaging* 35.5, pp. 1299–1312.

Ting, Daniel Shu Wei et al. (2017). “Development and validation of a deep learning system for diabetic retinopathy and related eye diseases using retinal images from multiethnic populations with diabetes”. In: *JAMA - Journal of the American Medical Association* 318.22, pp. 2211–2223.

torch (2019). *PyTorch - torchvision.transforms: Normalization function.* [Online accessed 2019-01-12]. URL: `https://pytorch.org/docs/stable/torchvision/transforms.html`.

Torre, Jordi de la, Domenec Puig, and Aida Valls (2018). “Weighted kappa loss function for multi-class classification of ordinal data in deep learning”. In: *Pattern Recognition Letters* 105, pp. 144–154.

Voets, Mike, Kajsa Møllersen, and Lars Ailo Bongo (2018). “Replication study: Development and validation of deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs”. In: *arXiv preprint arXiv:1803.04337*.

Wang, Zhiguang and Jianbo Yang (2017). “Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation”. In: *arXiv preprint arXiv:1703.10757*.

Wickham, Hadley (2016). *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. ISBN: 978-3-319-24277-4. URL: `http://ggplot2.org`.

Xu, Jun and Johan Dunaventand Raghu Kainkaryiam (2015). *Summary of our Solution to the Kaggle Diabetic Retinopathy DetectionCompetition.* github. URL: `https://storage.googleapis.com/kaggle-forum-message-attachments/88866/2815/Team_Reformed_Gamblers_Solution_Summary_v2.pdf`.

Zhang, Jian and Ioannis Mitliagkas (2017). “Yellowfin and the art of momentum tuning”. In: *arXiv preprint arXiv:1706.03471*.

Zheng, Yingfeng, Mingguang He, and Nathan Congdon (2012). “The worldwide epidemic of diabetic retinopathy”. In: *Indian journal of ophthalmology* 60.5, p. 428.

Figures

Figure 18: Progress of accuracy and Cohen's kappa score with less than 1000 images used for training the neural network.

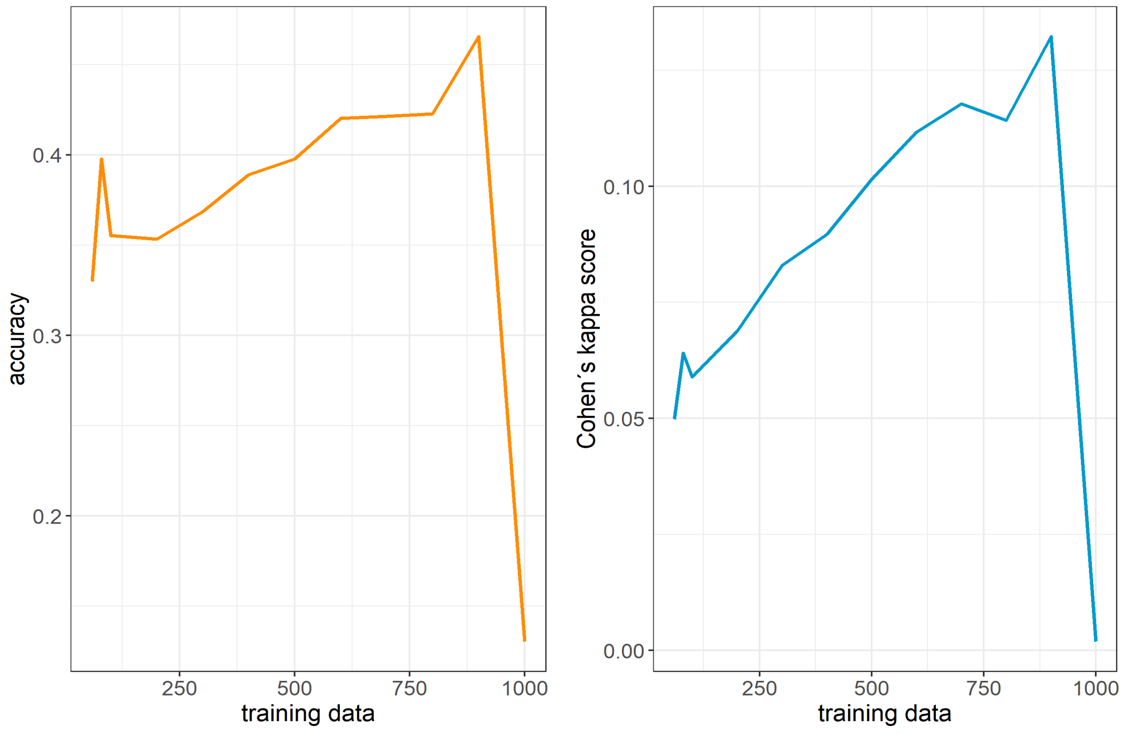
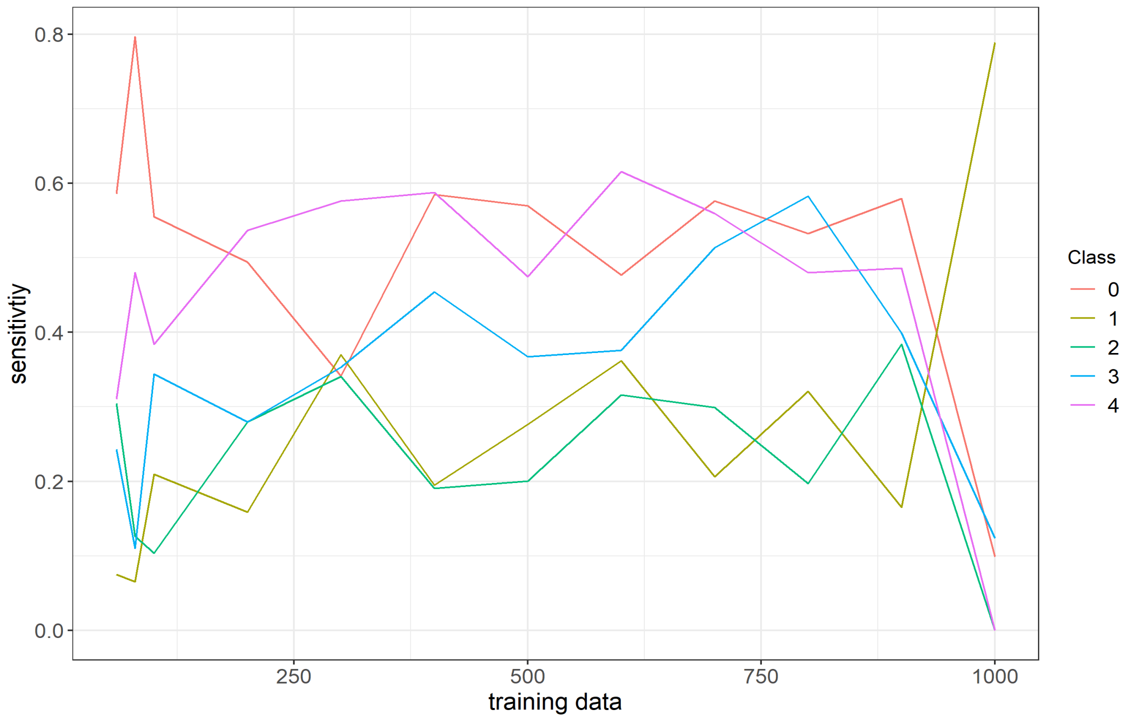


Figure 19: Progression of the Sensitivity per class with less than 1000 training images.



Tables

Table 11: Extract of the settings used to train neural networks on a server.

Parameter	Loop0	Loop1	Loop2
architecture	inception_v3	inception_v3	inception_v3
balancing	False	False	False
batch_size	2048	2048	2048
bootstrap_iters	50	50	50
cost_term	False	False	False
data_dir	Data/IMG_resized	Data/IMG_resized	Data/IMG_resized
distortion	True	True	True
epochs	100	100	100
exponent	2	2	2
factor_lr	0.1	0.1	0.1
fp16	True	True	True
freeze_all_layers	False	False	False
freeze_sub_layers	True	True	True
greyscale	False	False	False
hist_equalization	False	False	False
image_size	299	299	299
learning_rate	0.001	0.001	0.001
loss	CrossEntropy	CrossEntropy	CrossEntropy
max_bad_runs	8	8	8
n_augm	60000	60000	60000
n_test	all	all	all
n_train	29270	26000	23000
normalize	True	True	True
nr_sub_layers	15	15	15
online_augm	True	True	True
oversampling	False	False	False
parallel	True	True	True
rotate	True	True	True
scale	True	True	True
shear	True	True	True
skew	False	False	False
step_size	12	12	12
use_augm	False	False	False
weights	True	True	True
zoom	False	False	False

Table 12: Confusion Matrix with totals per class. This is for the test data set when training is done with all images.

		Predicted					
	class	0	1	2	3	4	total
	0	5581	72	764	7	28	6452
	1	472	23	115	0	1	611
True	2	587	21	655	38	22	1323
	3	26	0	119	65	8	218
	4	17	1	72	14	73	177
	total	6683	117	1725	124	132	8781

Table 13: Confusion Matrix with ratio of predicted images per class per true class if all images are used to train the neural network.

		Predicted				
	class	0	1	2	3	4
	0	0.86	0.01	0.12	0.00	0.00
	1	0.77	0.04	0.19	0.00	0.00
True	2	0.44	0.02	0.50	0.03	0.02
	3	0.12	0.00	0.55	0.30	0.04
	4	0.10	0.01	0.41	0.08	0.41

Table 14: Bootstrap Iterations depending on the number of overall images n in training process

Nr. training images	iterations
$0 < n \leq 500$	10
$500 < n \leq 1000$	6
$1000 < n \leq 10000$	1
$10000 < n$	1

Readme

Data

The used data set is from a kaggle challenge back in 2015 (challenge-link). This dataset can be downloaded either using the kaggle api (api documnetation) or using the lrz lync+share folder (lrz). Permission to the folder can only be given to members of the lrz network.

In the electronical appendix of the thesis a folder exists where all images are stored. These images are already resized and hist equalization is applied. These can be used. But for more flexibility in the preprocessing steps original data has to be used.

Results

All results are stored in the results folder of the electronical appendix. In the thesis not all results are used, but in the appendix all results are stored. So it is possible to even have a deeper look into the results. The analysis with the scripts in 'Python/-AnalysisThesis/Analysis_results_week.R' are usable if the folder structure stays the same.

Preprocessing of Data

All scripts for preprocessing and training the neural networks are stored in the Python folder. If the folder structure stays the same, these are usable as they are right now.

The code to train a neural network is written in python. The framework pytorch is used. The pytorch framework uses the functions called ImageFolder and Dataloader for loading data into the framework. This function uses the name of the subfolders as labels for the data. So the images have to be distributed to the five folders with the names of the classes.

Steps to get this done:

1. Set up folder called Data with subfolder called train where all images are stored.
2. In the folder Data needs to be the csv file with the labels for each image called 'trainLabels.csv'
3. Run 'readingData.py'. First the images are resized to the dimension (512, 512). Then the images are distributed by label. Path and size can be adapted. Images are stored into 'Data/IMG_resized'.

Optional:

- For better results hist equalization is used. For this run file ‘histEqualization.py’. Either a classical hist equalization can be done or a contrast limited adaptive hist equalization can be used. Change parameter and function in line 58.

Train neural network

All other data preprocessing, training and testing steps of training a neural network are automized. The file ‘main.py’ combines all settings. Steps to train a network:

1. In the folder ‘Python/Settings’ are some example settings files. Open ‘settings.csv’ with Excel.
2. Change to preferred settings. Some constraints and hints on that:
    - First Row so naming of columns need to be Loop with ascending numbers starting from 0: ‘Loop0’, ‘Loop1’, . . .
    - oversampling can only be used with balancing. Either way, here should be considered that for really unbalanced sets images in smaller classes are multiple times duplicated. Better use ‘use_augm’. Here Data Augmentation is used and classes are automatically balanced
    - n_augm >= n_train with use_augm = True
    - batch_size < n_augm for use_augm = True or batch_size < n_train
    - ‘parallel’ can only be used when multiple gpus are available
    - ‘fp16’ is suggested to be only used when parallel is used.
    - loss can be either CrossEntropy or MSE. There is a possibility to use a weighted kappa loss ‘wkappa’, but this is not stable, neither checked or recommended.
    - exponent is only used when cost_term = True or for loss = wkappa.
    - online_augm and use_augm cannot be True at the same time.
    - On cpu batch_size should not be higher than 256, use smaller batch and train sizes. This is because of memory limitations.
    - For training sizes smaller than 10000 images bootstrapping and cross validation is going to be used.
3. Change line 61 of ‘main.py’ file to the correct name of the settings file.

4. Run ‘main.py’ out of the folder consisting Python and Data folders. Use ‘python Python/main.py’.

A lot of other parameters can be changed but will only be changed in the python files. For that see the different defined functions.

Always consider what impact a change will result in, because one can have major influence on run time and memory consumption.

Possible Errors

Here some possible errors are presented that are occuring sometimes:

- batch size and number training images does not fit always. A mini-batch cannot have only one image. This is sometimes occuring especially with smaller batch sizes. No automatic solution is included. Manual changes have to be made.

Scripts

Further details of the functions are in the function description in the files. Included are details about the parameters.

Here only the rough algorithm structure of the scirpts or functions is presented

1. main.py

In this file the settings are loaded and based on these all other steps are initiated. Main differentiation are the split of the data into training, validation and test set. Particularly how many images are selected for training the networks. Also it is differentiated if cross validation is used or not.

Listing 1: main.py

```
Read settings file
loops = set nr of loops
for i in range(loops):
        if nr training data <= 1000:
            straps = set bootstrapp iterations
            for bs_iter in range(straps):
                split data into training, valdiation & test set
                   using split_data()
                run cv_function
                run clean_data()
                save results of single bootstrap iteration into
                   lists
            calculate cv results & save everything
        else:
            split data into training, validation & test set using
                split_data()
            run transfer_learning()
            run clean_data()
            save results
```

2. split_data() in distributeData.py or distributeData_2.py

This function distributes the data into training, validation and test sets. Two versions are available:

i. First distributes the data as uniformly as possible per class:

Listing 2: split_data() (Version 1)

```
Check & print if there are images missing in any class
Initialize group size DF with available images per class & data
    set
labels = classes
if n_train_all == 'all':
        for lab in labels:
            set nr images per class & data set
            Write into group size DF
else:
        for lab in labels:
            set nr images per class & data_set
            if n_val_group == 0:
                add 1
if not n_train_all == 'all':
        while sum of trainig and validation images < n_train_all:
            fill images from other classes where nr of images <
                n_train_group
for lab in labels:
        read image names
        sample image names
        write selected nr of images into train, val & test set
```

ii. Second distributes the data with the original distribution per class:

Listing 3: split_data() (Version2)

```
Check & print if there are images missing in any class
```

```
Initialize group size DF with available images per class & data
    set
labels = classes
Calculate ratio per class
for lab in labels:
        calculate nr of images for training, validation & test
            set per class based on ratios
        Handle expections if nr of training or validation data is
            0
for lab in labels:
        read image names
        sample image names
        write selected nr of images into train, val & test set
```

3. cv_function() in cross_validation_function.py

Cross validation splits the data into six equal sized parts and uses successively one of the as validation set and the rest as training data. So after the initial split into training, validation and test set, in this function the data of training and validation set is first merged again. This data is randomly splitted into the six different parts. A loop over these parts is used and the neural network is trained six times using the different combinations. The best performing network is used as the ouptut.

Listing 4: cv_function()

```
for lab in classes:
        merge training & validation set
select cross validation algorithm
for train_index, val_index in cv_split_algo:
        # Algorithm sets training and validation index
            automatically
        distribute iamges to training & validation folders
        run transfer_learning()
        if not last cv iteration:
            distribute data back to merged folder
        if kappa of last run > best kappa:
            store new results
return cross validation results
```

4. transfer_learning() in transfer_learning.py

This function includes data processing, transfer learning, training and testing the model. So this function assembles all main parts of training a neural network.

Listing 5: transfer_learning()

```
set image_size based on selected architecture
run data_preprocessing()
load pretrained_model
freeze selected nr of layers
Replace classification layer
Calculate weights for weighted loss function
Set loss function
set optimizer
run train_model()
run test_model()
Write stats & results into stats file
return results
```

5. data_preprocessing() in dataPreprocessing_new.py

This function loads and processes the data. Offline and online augmentation is done here. Balancing can be done here. Output is dataloader that include finished preprocessed training, validation and test data.

Listing 6: data_preprocessing()

```
if normalize images:
        calculate mean & sd per pixel per & channel
        add to transform function
if online augmentation:
        add augmentation functions to transform function
read images of selected folder & apply transform function
if balancing:
        run balance_train()
if offline augmentation:
        run data_augmentation()
for set type:
        run Dataloader() from torch library
return dataloaders & dataset_sizes
```

6. balance_train() in dataPreprocessing_new.py

The dataset can be balanced. In this case in every class should be the same amount of data points. If a class has to many points this will be renamed to the original data folder. If a class does not have enough data, data augmentation is used to fill the void.

Listing 7: balance_train()

```
n_train_group = set nr training images per class
if balancing and not oversampling:
        calculate available data
        for lab in labels:
            if not offline augmentation and avail data <
               n_train_group and not oversampling:
                calculate diff between n_train_group & avail_data
                run data_augmentation() as balancing
            if avail_data > n_train_group:
                store dispensable images in unused folder
if balancing and oversampling:
        for lab in labels:
            copy random selected images to balance training set
```

7. data_augmentation() in dataAugmentation.py

Data Augmentation uses predefined techniques to augment the file and save this new image on the disk.

Listing 8: data_augmentation()

```
Calculate nr of augmented images depending on total images & that
    not more than 10 x of the original images should be augmented
for i in classes:
        set nr augmented images for class i
        set up Augmentor Pipeline
        Activate augmentation techniques depending on input
           settings
        Create augmented images
        Rename images into correct folder
```

8. train_model() in trainModel.py

Training a neural network includes two phases: Training and validation. In the training phase the outputs of the images in a mini-batch are calculated. Then the loss is calculated and backward propagation is used to adapt the weights. Also the learning rate is adapted using a step size learner. In the validation part the outputs are calculated and then coefficients are calculated. The early stopping theorem is applied based on the validation data results.

Listing 9: train_model()

```
Initiate best coefficient values
for epoch in range(num_epochs):
        for phase in ('train', 'val'):
            if phase == 'train':
                Adapt learning rate with step learner
            else:
                model to evaluation mode
            for input, labels in dataloader:
                zero parameter gradients
                calculate model outputs
                if loss == 'MSE':
                    calculate differences between output &
                       classes
                calculate top-2 predictions
                Calculate loss per class
                if cost_term:
                    Add cost term to output per class
                Average loss
                Store predictions
                if phase == 'train':
                    backward propagation
                    optimization step
            Calculate epoch loss & coefficients
            if phase == 'val':
                if epoch_kappa > best_kappa:
                    copy weights
                    set bad_runs to 0
                else:
                    add 1 to bad_runs
        if bad_runs == max_bad_runs:
            early stopping
store stats & model with best weights
return models & stats
```

9. test_model() in trainModel.py

The outputs of the test data images is calculated and the coefficients are calculated.

Listing 10: test_model()

```
set model to evaluation mode
for inputs, labels in test_data:
        calculate outputs
        calculate top-2 predictions
        calculate loss per class
        avearge loss
calculate loss & coefficients
write stats
combine labels & predictions in file
Write Confusion Matrix & sensitivities
return stats & other results
```

10. clean_data() in transfer_learning.py

The images are renamed back into their initial lab folders.

Listing 11: clean_data()

```
for lab in classes:
        list all training files
        remove augmented images
for set in ('train', 'val', 'test'):
        for lab in classes:
            rename unused images back to actual folders
```

Comments

- To change which version of the split_data function is used, the source of the function in line 22 in 'main.py' has to be changed. Possible sources are:
  i. distributeData
  ii. distributeData_2
- Generally run all scripts out of the base folder meaning the folder including the Data and Python folder.
- There exists a script to transform the data into a binary classificiation problem. This script is called '2classes.py'. After running the script, the code should work the same.